Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ

Кафедра промышленной электроники

К. В. Бородин

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА И СИСТЕМЫ

Методические указания по выполнению лабораторных работ

Томск 2016

Корректор: А. Н. Миронова

Бородин К. В.
Микропроцессорные устройства и системы : методические указания по выполнению лабораторных работ. – Томск : ФДО, ТУСУР, 2016. – 64 с.

Пособие служит руководством по выполнению лабораторных работ по курсу «Микропроцессорные устройства и системы». Цель работ: усвоение основных принципов работы с программами эмуляции и отладки устройств на микроконтроллере фирмы AVR. Изучаются программная модель, система команд и характеристики периферийных устройств микроконтроллеров ATmega16 с использованием программного продукта Atmel Studio 6.2 для отладки программ на ассемблере и языке Си.

Для студентов вузов радиоэлектронного профиля и инженеров-проектировщиков средств и систем автоматики и промышленной электроники.

ОГЛАВЛЕНИЕ

1 ПРОГРАММНЫЙ ПАКЕТ ATMEL STUDIO

Atmel Studio – интегрированная среда разработки (IDE) от компании Atmel для разработки приложений под микроконтроллеры ARM Cortex-M и AVR. Ранее назывался AVR STUDIO. В данном пособии приведены экраны программы версии Atmel Studio 6.2. Распространяется свободно с официального сайта компании Atmel.

Программный продукт AVR Studio разрабатывается с 2004 г. Программа позволяет работать как на ассемблере, так и на C/C++, содержит в себе:

- интегрированный компилятор C/C++;
- интегрированный симулятор;
- отладчик (Debugger);
- мастер проектов;
- обширную онлайн-базу готовых примеров;
- поддержка инструментов Atmel, совместимых с 8-разрядной AVR архитектурой, в том числе AVR ONE!, JTAGICE mkI, JTAGICE mkII, AVR Dragon, AVRISP, AVR ISPmkII, AVR Butterfly, STK500 и STK600;
- поддержка плагина AVR RTOS (операционная система реального времени);
- поддержка AT90PWM1 и ATtiny40;
- программное обеспечение верхнего уровня для поддержки внутрисхемного программирования (In-System Programming, ISP).

Визуальные инструменты позволяют ускорить написание программы. Благодаря связке программных пакетов Atmel Studio и Proteus от фирмы Labcenter Electronics возможно программирование микроконтроллеров без наличия какой-либо материальной базы.

Отладчик AVR Studio имеет два режима работы: режим программной симуляции и режим управления различными типами внутрисхемных

эмуляторов (In-Circuit Emulators) производства фирмы Atmel. Важно отметить, что интерфейс пользователя не изменяется в зависимости от выбранного режима отладки.

После запуска AVR Studio для создания нового проекта необходимо в меню **Project** выбрать команду **New Project**. В результате на экране появляется диалоговое окно (рис. 1), в котором необходимо ввести название проекта (**Project name**) и его расположение (**Location**). Новый проект удобнее создавать в отдельной папке.

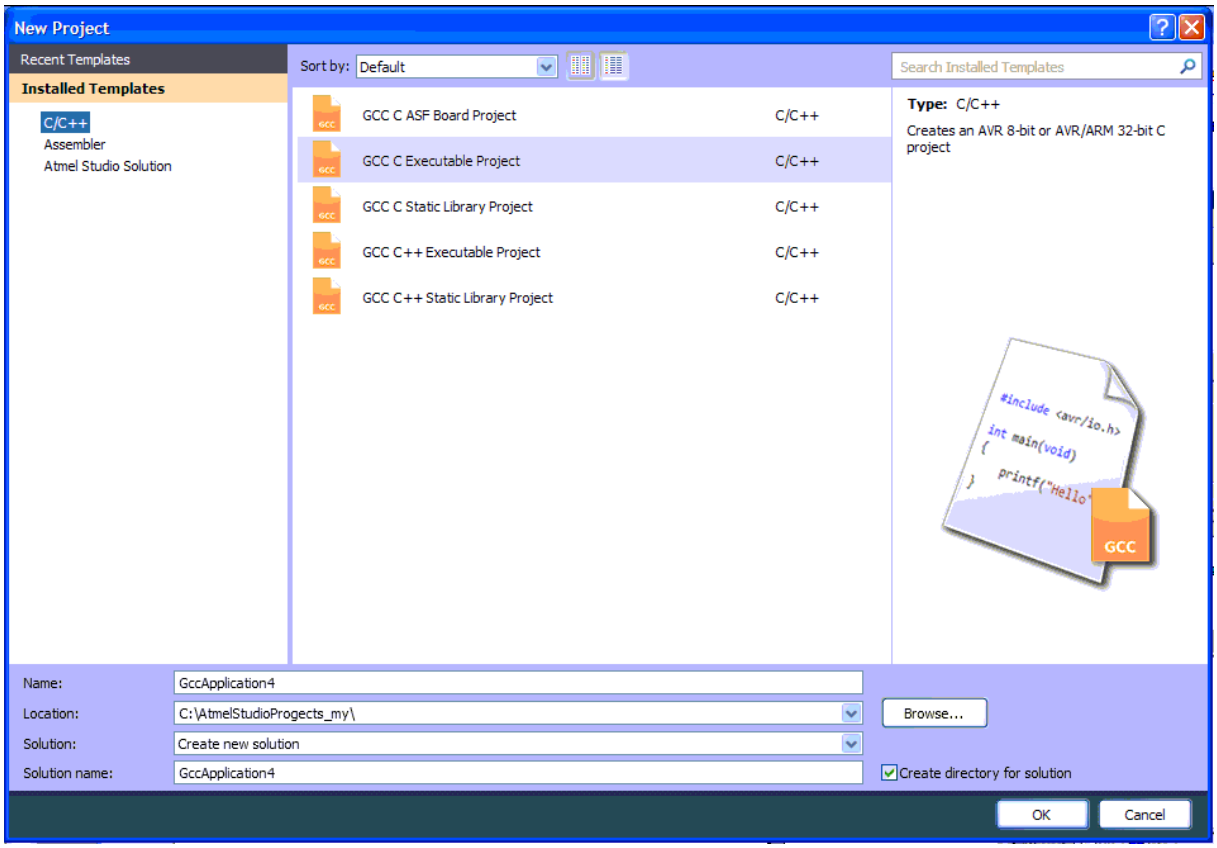

Рис. 1 – Окно создания нового проекта

Далее выбирается тип микроконтроллера. После нажатия кнопки **Finish** на экране появляется окно организации проекта (рис. 2), показывающее все связанные с проектом файлы, и окно для редактирования программы.

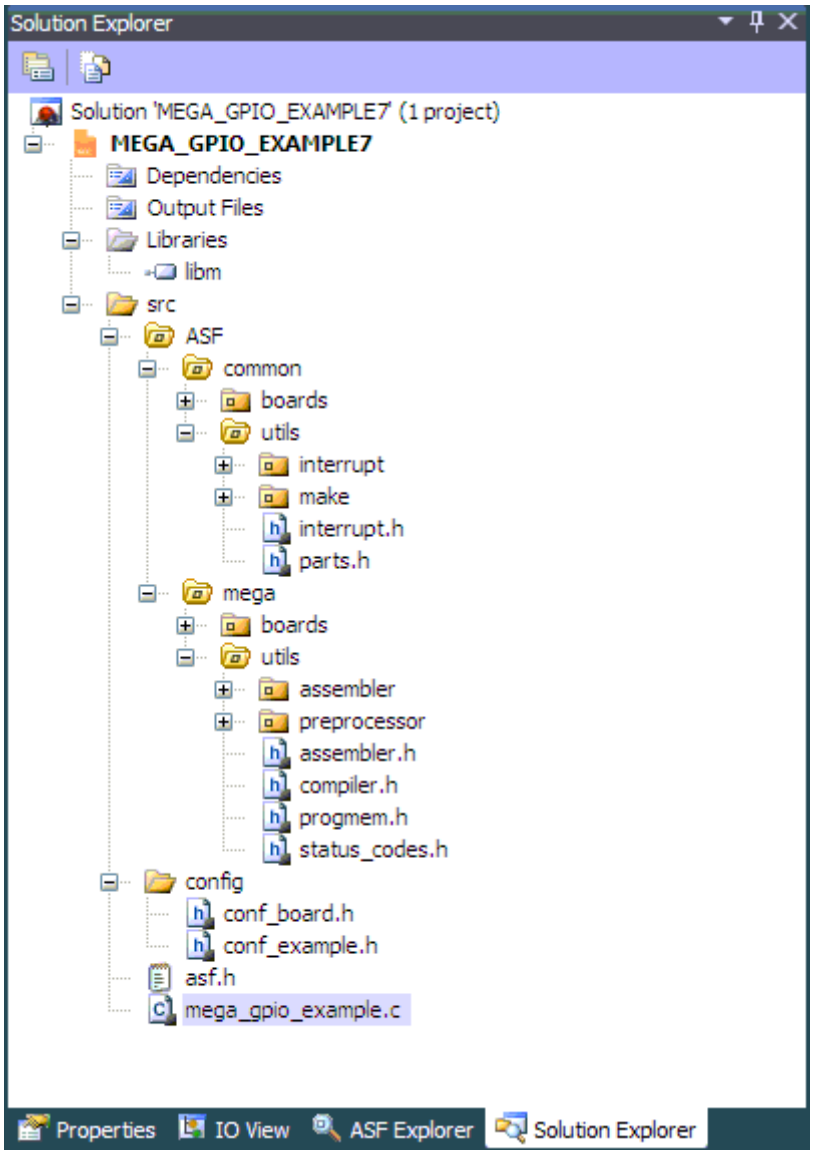

Рис. 2 – Окно организации проекта

В это окно для редактирования файла можно с клавиатуры ввести текст программы на языке ассемблера или открыть уже существующий файл (пункт **Add Fail** в меню **Project**) (рис. 3).

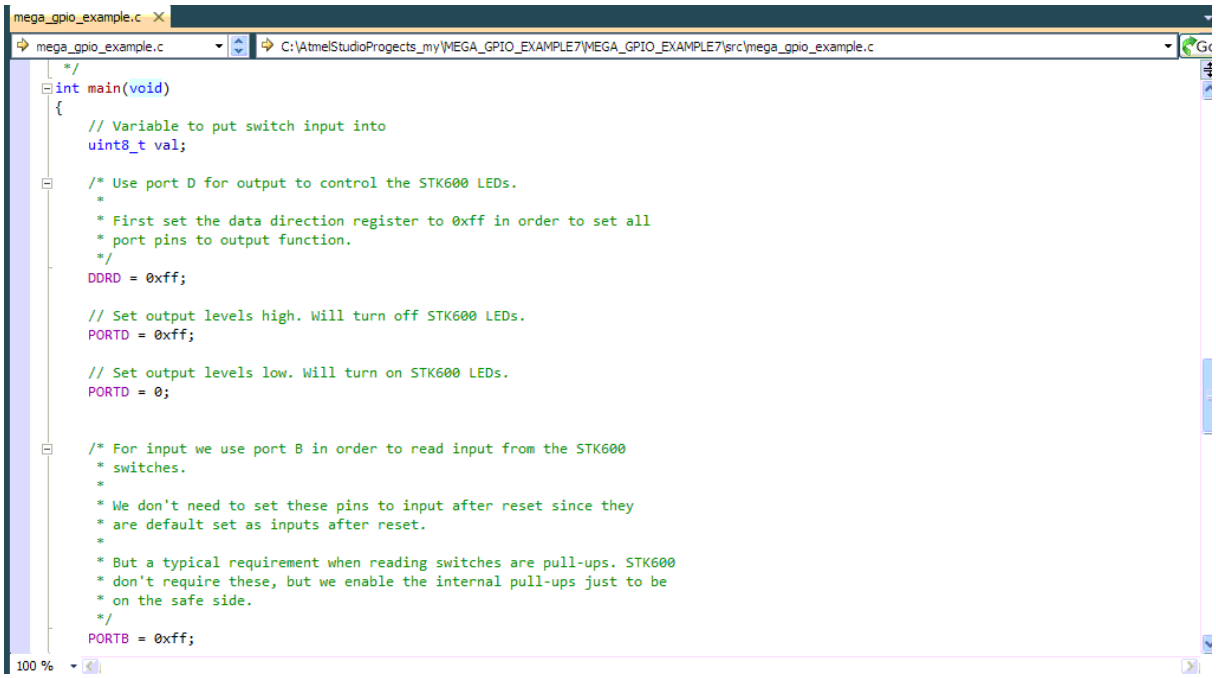

Рис. 3 – Окно редактирования программы на языке С

Все файлы проекта должны быть включены во входной файл проекта с помощью ассемблерной директивы .include. Для смены входного файла проекта на другой надо установить курсор мыши на нужный файл в окне организации проекта и щелкнуть правой кнопкой мыши.

Для осуществления трансляции программы и проверки правильности её написания выбирается пункт **Build** (кнопка **F7**) в меню **Project** (рис. 4).

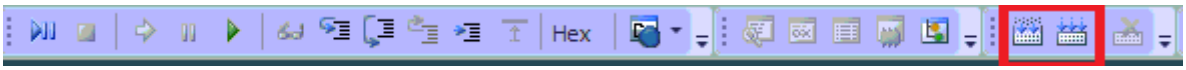

Рис. 4 – Панель компилятора

Окно **View Output** содержит сообщения компилятора. В это окно выводится информация о количестве слов кода и данных, о наличии ошибок и другая информация (рис. 5). Выводится информация, сколько процентов использовано в памяти программ (основной код программы) и памяти данных.

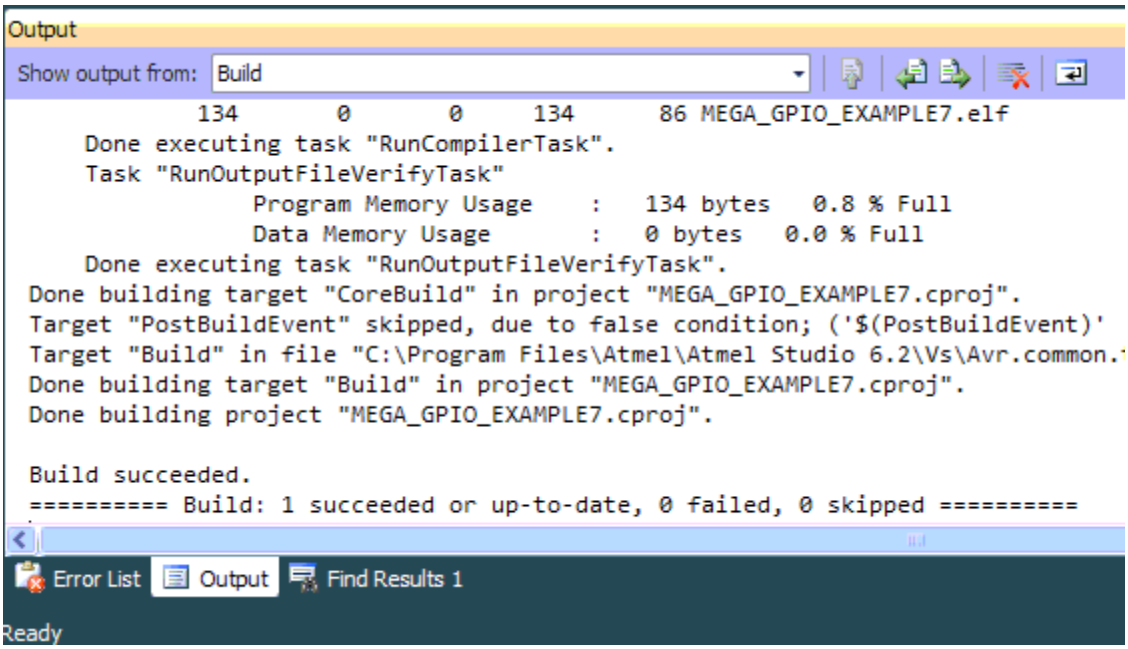

Рис. 5 – Окно сообщений компилятора

Для локализации ошибок трансляции в случае их наличия можно в окне сообщений установить курсор мыши на сообщение об ошибке и два раза щелкнуть левой кнопкой мыши. При этом в окне редактирования исходного текста программы курсор будет установлен на строку, вызвавшую сообщение об ошибке, и эта строка будет выделена цветом.

В результате трансляции создается выходной файл в указанном формате.

Для запуска отладчика необходимо выполнить процедуру **Build and Run**, которая вызывается при нажатии на соответствующую кнопку **(F7+Ctrl)** на панели управления. Процедура **Build and Run** выполняется в два этапа. Сперва происходит трансляция входного файла проекта, при которой независимо от установок проекта, кроме выходного файла заданного формата, генерируется и объектный файл. Затем этот объектный файл загружается в отладчик.

Экран AVR Studio в режиме отладки представлен на рис. 6.

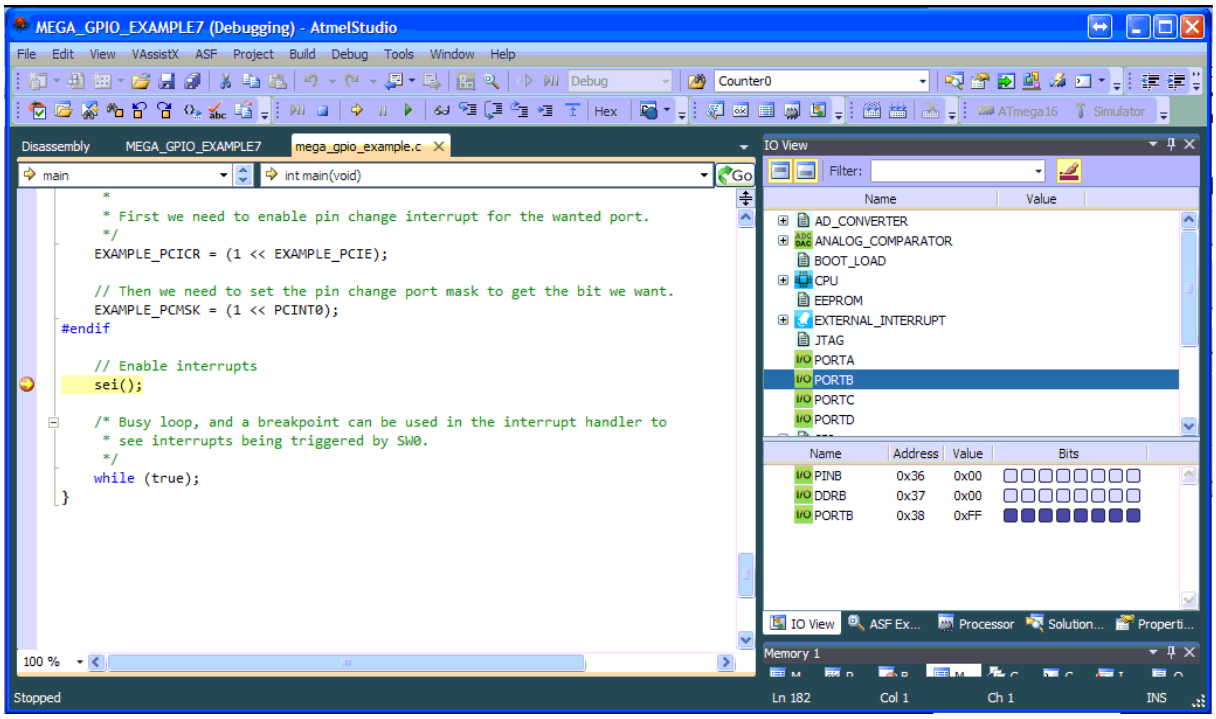

Рис. 6 – Экран Atmel Studio в режиме отладки

При выполнении процедуры **Build and run** (или при загрузке объектного файла) автоматически открывается окно исходного текста исполняемой микроконтроллером программы.

После выполнения процедуры появляется желтая стрелка, указывающая позицию программного счетчика микроконтроллера (рис. 7). Этот указатель всегда находится на строке, которая будет выполнена в следующем цикле.

Пользователь может выполнять программу полностью в пошаговом режиме, трассируя блоки функций или выполняя программу до того места, где стоит курсор. В дополнение можно определять неограниченное число точек останова, каждая из которых может быть включена или выключена. Точки останова сохраняются между сессиями работы.

В Atmel Studio для отладки программы предусмотрены две команды пошагового режима: **Step Over** и **Step Into**. Разница между ними в том, что команда Step Over не работает в подпрограммах. С помощью команд

пошагового режима можно проследить изменения значений в регистрах устройств ввода/вывода, памяти и регистрового файла. К командам шагового режима относятся также **Auto Step** и **Multi Step**. Помимо шагового режима возможна отладка программы с использованием точек останова (**Breakpoints**). Командой **Go** запускается исполнение программы. Программа будет выполняться до остановки пользователем или до обнаружения точки останова.

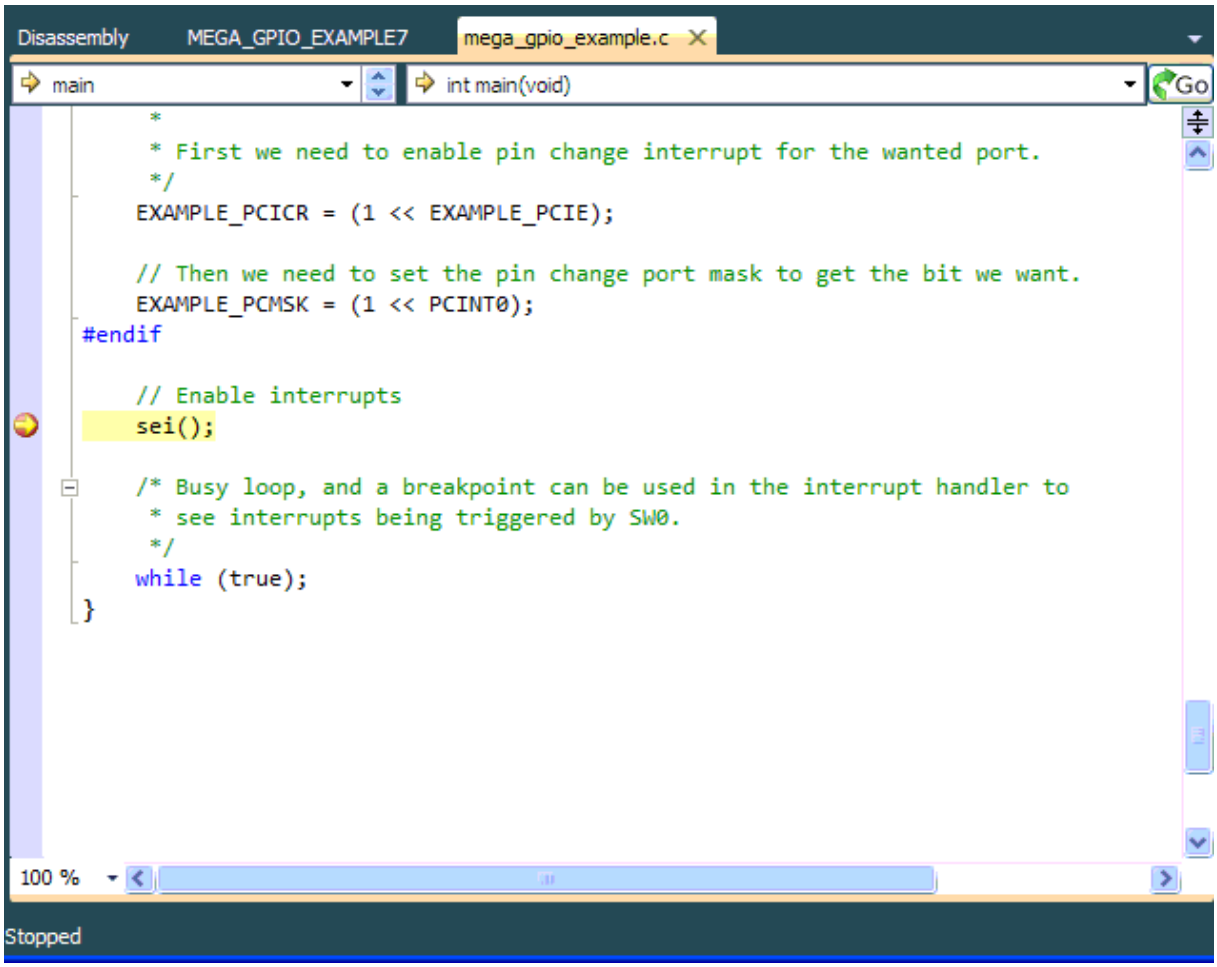

Рис. 7 – Окно исходного текста программы в режиме отладки

Для установки точки останова в Atmel Studio служит пункт меню **Debug -> Toggle Breakpoint**. Точка останова ставится в строке, отмеченной курсором. Красная отметка в левом поле окна исходного текста программы показывает установленную точку останова.

Имеется возможность отлаживать транслированный ассемблерный код программы. Для этого в окне текста нужно выбрать вкладку Disassembly (рис. 8).

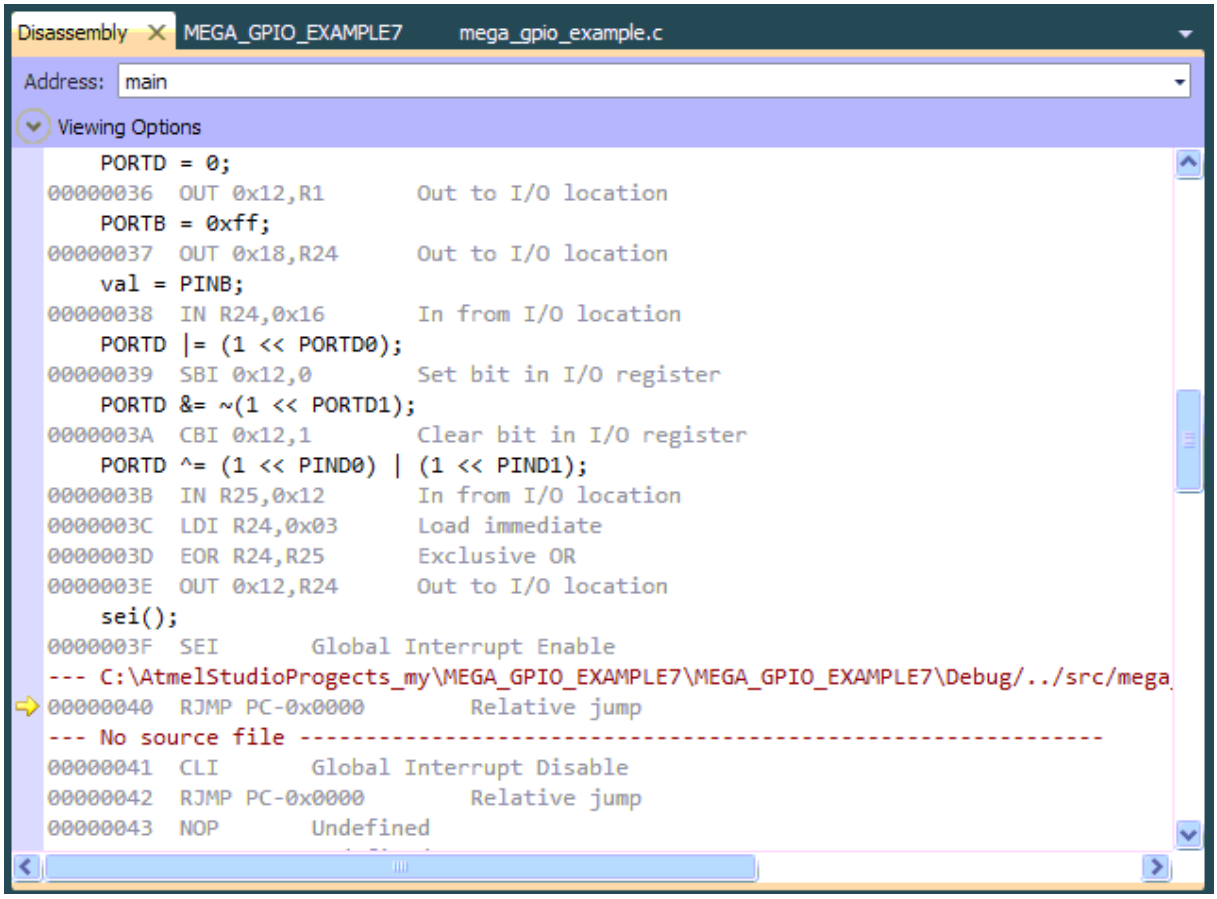

Рис. 8 – Окно дизассемблера с программным кодом

В процессе отладки также можно выбрать пункт меню **Debug -> Run To Cursor (Ctrl+F10)**. При выборе этого пункта исполняемый код выполняется до достижения команды, обозначенной курсором. При этом если отладчик обнаруживает точку останова, установленную ранее положения курсора, то останов будет выполнен только в случае его разрешения в окне **Debug Option**, в противном случае выполнение не приостанавливается. Если команда, обозначенная курсором, не достигается, отладчик продолжает исполнять код программы до тех пор, пока исполнение не будет

прервано пользователем. Поскольку режим **Run To Cursor** зависит от позиции курсора, он доступен только при активном окне исходного текста.

Для остановки исполнения программы пользователем служит команда **Break (Ctrl+F5)**. В состоянии останова эта команда недоступна. При отладке с использованием точек останова, или если адрес останова указан курсором в окне исходного текста, модификация информации во всех окнах происходит только при достижении останова (или при прекращении исполнения программы пользователем).

Пункт меню **Debug -> Reset (Shift+F5)** выполняет сброс микроконтроллера. Если программа при этом выполняется, то ее исполнение будет остановлено. После сброса информация во всех окнах модифицируется.

Для наблюдения за работой программы можно открыть несколько окон, отображающих состояние различных узлов микроконтроллера. Окна открываются нажатием соответствующих кнопок на панели инструментов или при выборе соответствующего пункта меню **View**.

Регистровый файл микроконтроллера AVR отображается в окне **Work space** (вкладка I/O, рис. 9), а также можно открыть отдельное окно **Registers** (рис. 9) Если в процессе выполнения программы в очередном цикле значение какого-либо регистра изменится, то этот регистр будет выделен красным цветом. При этом если в следующем цикле значение регистра останется прежним, то цветовое выделение будет снято. Такое же цветовое выделение реализовано в окнах устройств ввода/вывода, памяти и переменных.

Также в окне I/O WorkSpace отображается состояние встроенных периферийных устройств микроконтроллера (рис. 10).

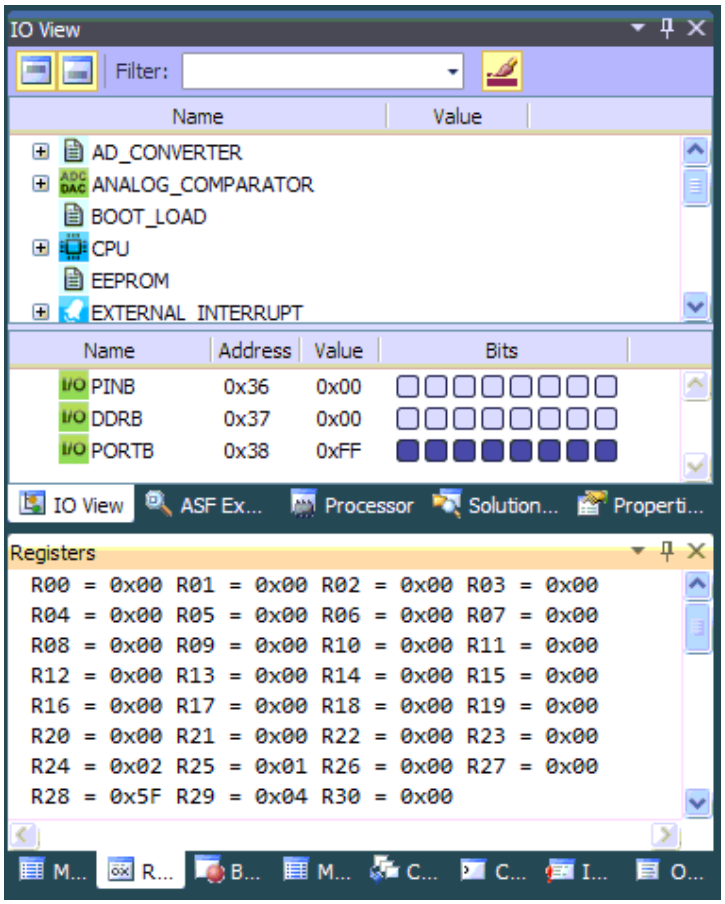

Рис. 9 – Окно состояния регистрового файла и окно I/O WorkSpace

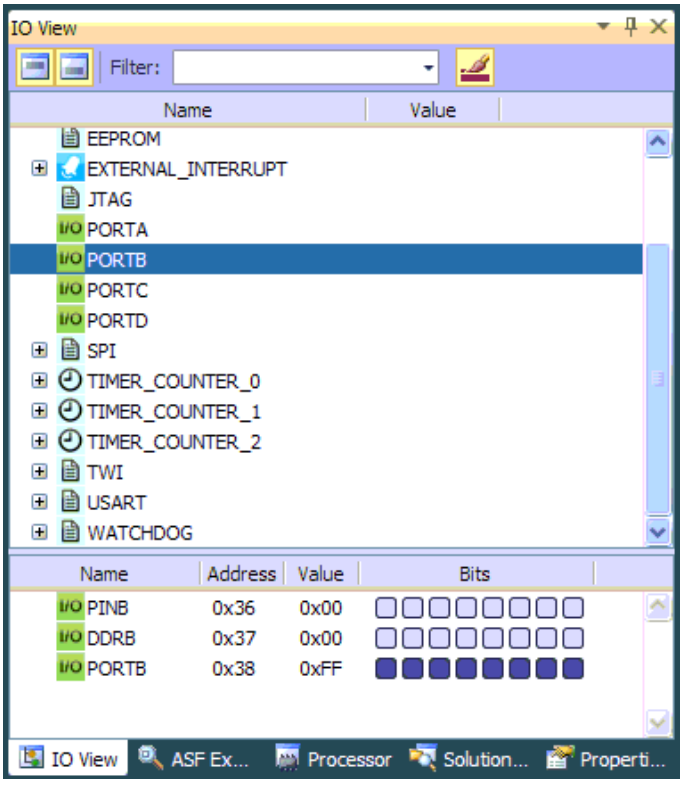

Рис. 10 – Развернутый порт PORTB и отображение
его регистров PINB, DDRB, PORTB

В этом окне отражаются все функциональные блоки *выбранного* микроконтроллера. Если выбрать другой контроллер, то часть функционала может отсутствовать. Любой блок может быть раскрыт нажатием на его значок. При раскрытии блока в окне отражаются адреса и состояния всех его регистров и отдельных, доступных для модификации, битов (рис. 11). Каждый доступный для модификации бит может быть установлен или сброшен как программой по ходу ее исполнения, так и пользователем вручную (указав курсором нужный бит и щелкнув левой кнопкой мыши, пользователь может изменить значение бита на обратное) – в режиме программной симуляции это является способом имитации входного воздействия на микроконтроллер. Например, имитируется нажатие кнопки или пришедшие данные в буфер UART, АЦП и др.

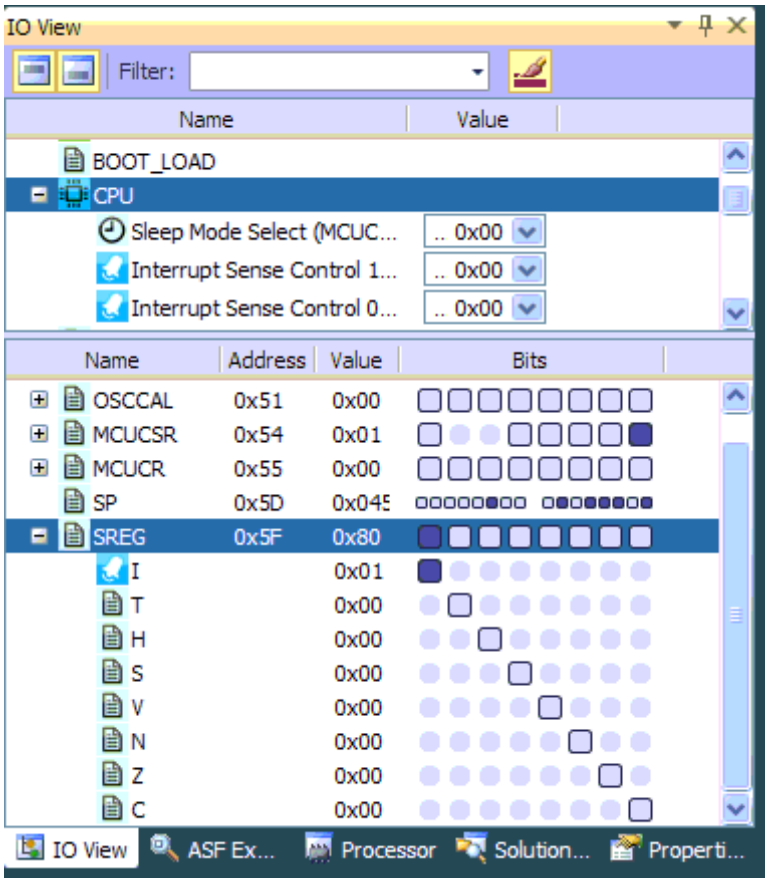

Рис. 11 – Окно состояния процессорного ядра

Для индикации состояния программного счетчика, указателя стека, содержимого регистра статуса SREG и индексных регистров X, Y и Z в процессе отладки программы предназначена вкладка **Processor** в окне I/O WorkSpace.

В этом же окне отображается текущее время выполнения программы и тактовая частота ядра микроконтроллера.

Просмотр ячеек памяти программ, памяти данных, EEPROM и регистров портов ввода/вывода в ходе исполнения программы осуществляется также с помощью диалогового окна **Memory**. Выпадающее меню диалогового окна позволяет выбрать один из четырех массивов ячеек памяти: Data, IO, Eeprom, Program Memory. Для одновременного просмотра нескольких областей окно **Memory** может быть открыто несколько раз. Информация в диалоговом окне может быть представлена в виде байтов или в виде слов в шестнадцатеричной системе счисления, а также в виде ASCII-символов (рис. 12).

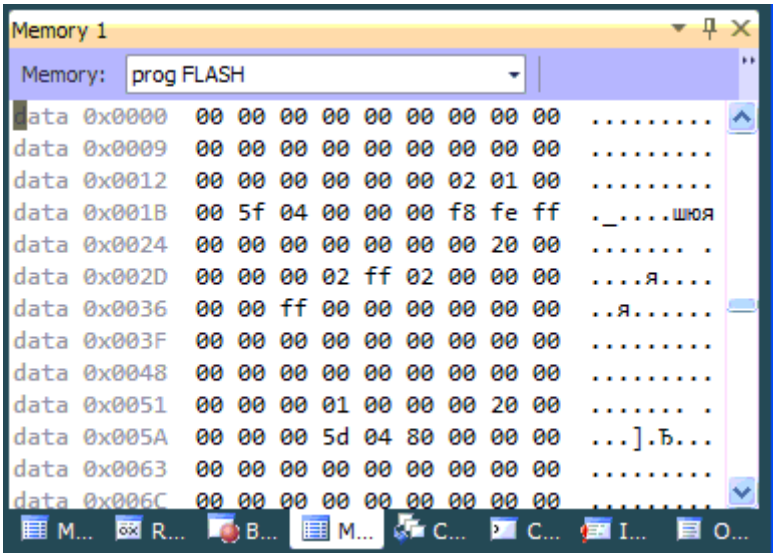

Рис. 12 – Окно просмотра содержимого памяти

Чтобы внести изменения в ячейке памяти, достаточно дважды щелкнуть мышкой по данной ячейке.

Для внесения изменений в программу в процессе отладки необходимо редактировать её исходный текст. При попытке запуска симулятора на исполнение программы после редактирования на экране появляется окно, сообщающее об изменении программы и необходимости её компиляции.

Для сохранения проекта необходимо воспользоваться пунктом меню **Project -> Close**. При закрытии проекта сохраняются все его настройки. Во время следующей загрузки настройки будут автоматически восстановлены.

2 ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Выбор варианта лабораторных работ осуществляется по общим правилам с использованием следующей формулы:

$$V = (N \times K) \text{ div } 100,$$

где V – искомый номер варианта,

N – общее количество вариантов,

div – целочисленное деление,

при $V = 0$ выбирается максимальный вариант,

K – код варианта.

2.1 Лабораторная работа № 1 «Порты ввода/вывода»

Цель работы

Целью лабораторной работы является изучение простейших команд языка С, портов ввода/вывода и отладка прикладных программ для микроконтроллера AVR семейства MEGA с помощью персонального компьютера и программного пакета Atmel Studio 6.2 (не ниже).

Краткая теория

Порты ввода/вывода AVR имеют число независимых линий «вход/выход» от 3 до 53. Каждая линия порта может быть запрограммирована на вход или на выход. Мощные выходные драйверы обеспечивают токовую нагрузочную способность 20 мА на линию порта (втекающий ток) при максимальном значении 40 мА, что позволяет, например, непосредственно подключать к микроконтроллеру светодиоды и биполярные транзисторы.

Общая токовая нагрузка на все линии одного порта не должна превышать 80–200 мА (в зависимости от корпуса и порта), а на все линии всех портов 400 мА (все значения приведены для напряжения питания 5 В).

Каждый порт обслуживают три регистра: регистр данных, регистр направления и регистр выводов.

Управление, настройка, контроль и другие всевозможные операции с выводами AVR-микроконтроллеров осуществляются всего лишь через три регистра:

– DDRx (направления);

– PORTx (настройки);

– PINx (состояния).

В работе Вам потребуется установить или сбросить бит/биты в разных регистрах. Сделать это можно с помощью логических операций «ИЛИ» и «И», используя их свойства – любое значение «ИЛИ» 1 равно 1, любое значение «И» 0 равно 0.

Например, для *установки* 3-го бита в регистре PORTA можно написать:

PORTA |= (1 << 3);

Данная запись означает для компилятора следующее:

1) константу – 0x01 (0b00000001) сдвинуть на 3 разряда влево. Получается 0x08 (0b00001000);

2) сделать операцию «ИЛИ» между значением регистра PORTA и результатом предыдущей операции;

3) записать результат предыдущей операции обратно в регистр PORTA.

Таким образом, если нам необходимо установить несколько бит в регистре, можно написать следующим образом:

PORTA |= (1 << 3)|(1 << 4)|…|(1 << 1);

Для *сброса* бита в регистре необходимо воспользоваться операцией «И» со значением, содержащим нули в интересующих нас позициях. Этого можно добиться с помощью операции инвертирования – «~».

PORTA &= ~(1 << 3);

PORTA &= ~((1 << 3)|(1 << 4)|…|(1 << 1));

Светодиодный семисегментный индикатор представляет собой группу светодиодов, расположенных в определенном порядке и объединенных конструктивно (рис. 13). Зажигая одновременно несколько светодиодов, можно формировать на индикаторе цифры либо некоторые буквы. Индикаторы различаются по типу соединения светодиодов – общий анод, общий катод; по количеству отображаемых разрядов – однораразрядные, двухразрядные и т. д.; по цвету – красные, зеленые, желтые и т. д.

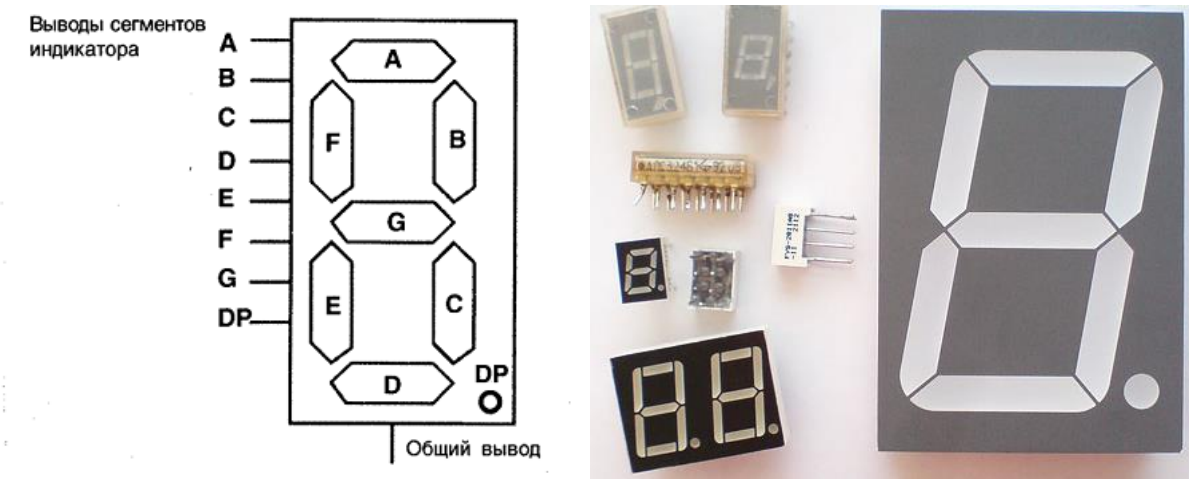

Рис. 13 – Обозначение и внешний вид семисегментных индикаторов

Чтобы зажечь на индикаторе какую-то цифру, нужно настроить порты, к которым подключен индикатор, на выход и установить в порту микроконтроллера код нужной цифры.

Пусть семисегментный индикатор подключен напрямую к порту (вывод A индикатора = младшему биту порта (0), вывод DP = старшему (7)). Тогда массив кодов цифр будет представлять:

```
        unsigned char number[] =
{
        0x3f, //0
        0x06, //1
        0x5b, //2
        0x4f, //3
        0x66, //4
        0x6d, //5
        0x7d, //6
        0x07, //7
        0x7f, //8
        0x6f //9
};
```

Используя десятичные цифры от 0 до 9 в качестве индекса, можно выводить в порт нужные коды командой.

PORTB = number[1]; //вывод числа 0x06 в порт, которое соответствует цифре 1.

Программа работы

Запустить программу Atmel Studio 6.2 (не ниже), ярлык которой расположен в меню Atmel -> Atmel Studio (рис. 14). В работе представлены экраны программы Atmel Studio 6.2.

Рис. 14 – Меню запуска программы Atmel Studio 6.2

После запуска откроется основное стартовое окно компилятора (рис. 15). Язык программы английский, как и других программ для программирования микроконтроллеров. На стартовой странице можно видеть ранее созданные проекты, часто задаваемые вопросы и примеры, в том числе видеопримеры. Знакомство с незнакомым программным продуктом лучше всего начать с примеров.

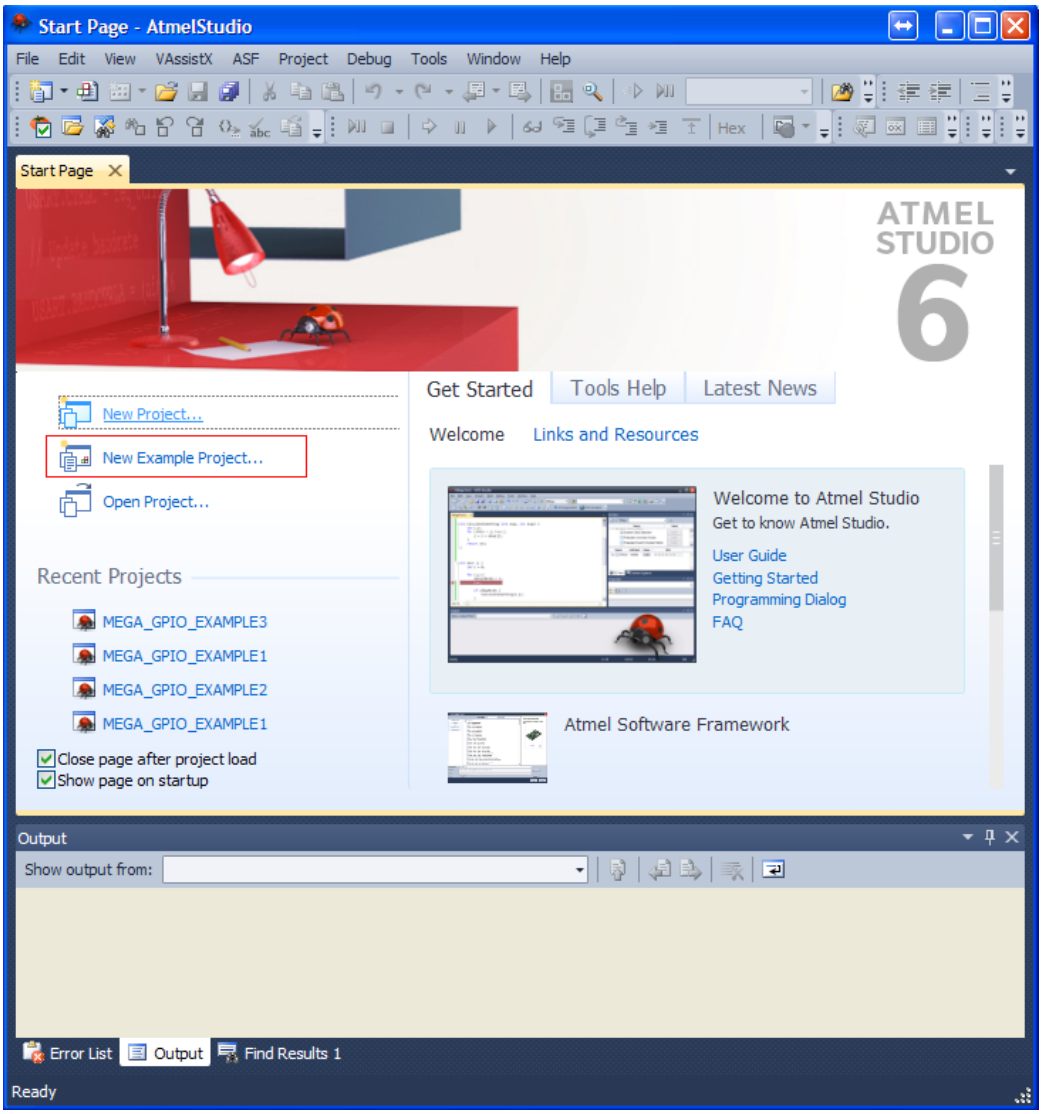

Рис. 15 – Стартовое окно программы

Выбрать **New Example project**. Откроется окно выбора примеров проектов (рис. 16). Данная база загружается с сайта ATMEL (требует интернет-соединения) и постоянно пополняется новыми примерами. В ней представлены различные отлаженные примеры отдельных блоков и рабочих больших комплексных проектов.

В верхней части окна выбрать готовый пример для ATmega16:

megaAVR, 8-bit, Applications , megaAVR GPIO example – STK600 – ATMEGA16.

В нижней части указать имя проекта и путь. Они не должны содержать русских символов. Нажать **ОК** при готовности.

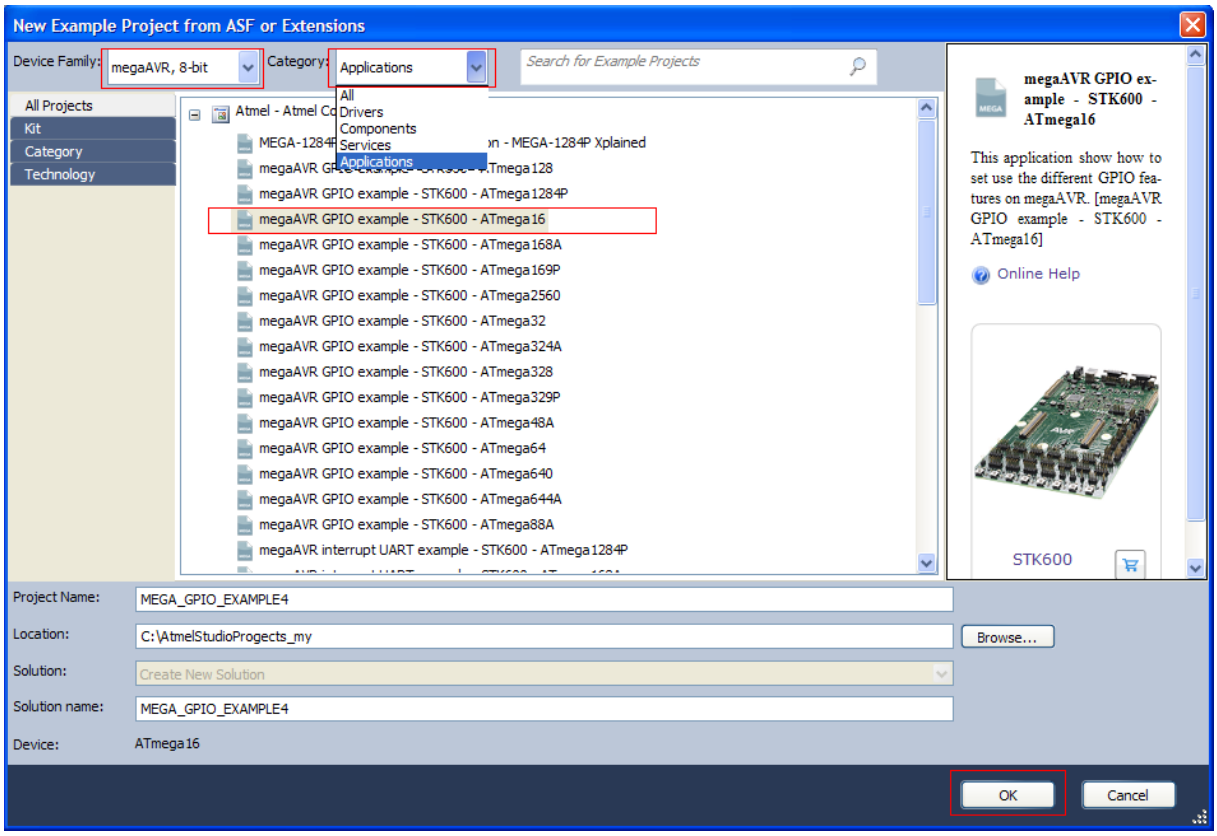

Рис. 16 – Окно выбора контроллера

Откроется выбранный пример использования портов GPIO (рис. 17).

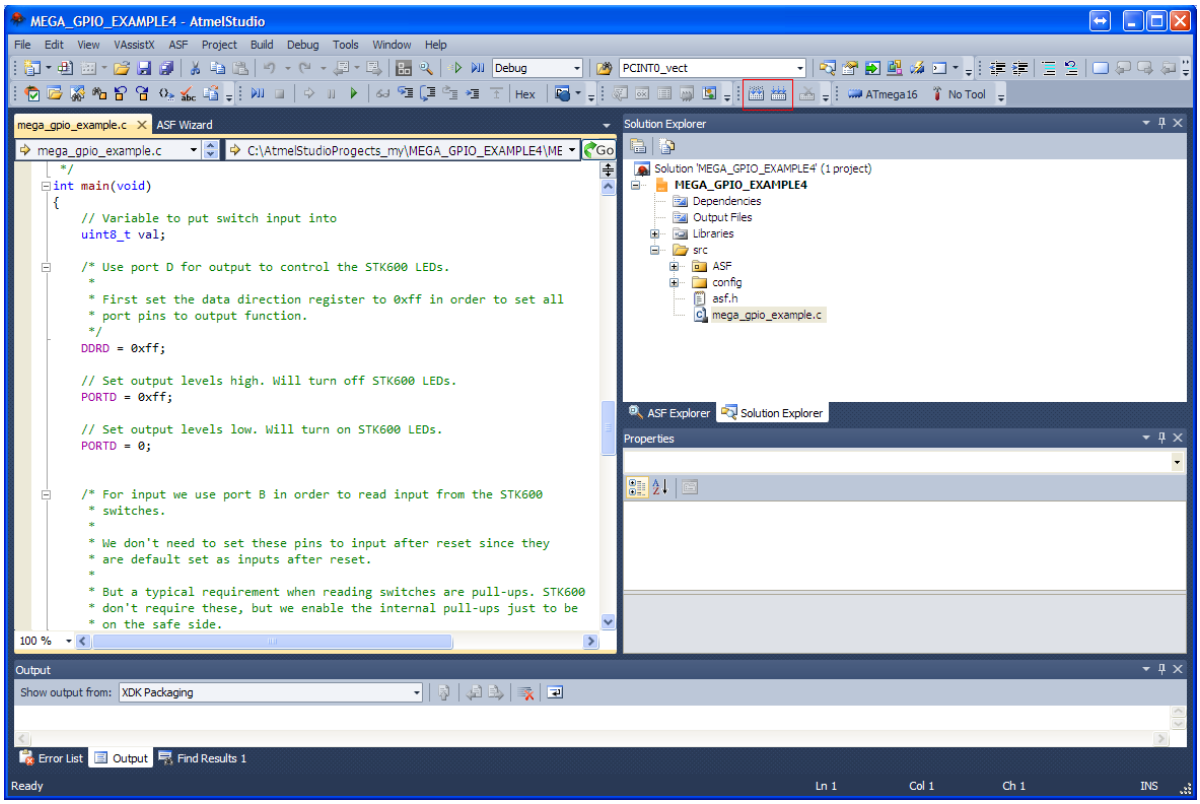

Рис. 17 – Основное рабочее окно программы

В выбранном примере показана обработка внешнего прерывания INT0 при нажатии кнопки. Работа с прерываниями рассматривается во второй и третьей лабораторных работах.

Задание 1. Знакомство с компилятором и режимом отладки (общее для всех вариантов)

Найти и удалить из проекта следующий код:

```
// Toggle both pin PD0 and PD1, which toggles LED0 and LED1.
#if defined(__AVR_ATmega16__) || defined(__AVR_ATmega32__) \
                || defined(__AVR_ATmega64__) || de-
fined(__AVR_ATmega128__)
/* For older megaAVR devices read-modify-write PORT register.
         * This isn't safe for interrupts.
         */
        PORTD ^= (1 << PIND0) | (1 << PIND1);
#else
        // Use PIN register to toggle on newer megaAVR devices
        PIND = (1 << PIND0) | (1 << PIND1);
#endif

// Only use Pin Change Interrupt handler for devices supporting this.
#ifdef EXAMPLE_PCICR
/* Enable pin change interrupt for PB0 which is controlled by SW0
         *
 * First we need to enable pin change interrupt for the wanted port.
         */
        EXAMPLE_PCICR = (1 << EXAMPLE_PCIE);

// Then we need to set the pin change port mask to get the bit we want.
        EXAMPLE_PCMSK = (1 << PCINT0);
#endif
        // Enable interrupts
        sei();

/* Busy loop, and a breakpoint can be used in the interrupt handler to
         * see interrupts being triggered by SW0.
         */
        while (true);
```

Заменить

PORTD &= ~(1 << PORTD1);

на

PORTD &= ~(1 << PORTD0);

Добавить в бесконечный цикл while(1)

```
while(1)
{
	/* It's also possible to control a single or a set of pins on a port.
	 *
	 * Setting pin PD0 high in order to turn off LED0.
	 */
	PORTD |= (1 << PORTD0);

	// Setting pin PD1 low in order to turn on LED1.
	PORTD &= ~(1 << PORTD0);
}
```

Проект готов для отладки и симуляции.

Нажать на кнопку **Build Solution (F7)**, что на панели компиляции (рис. 18), и **Старт (Start Debugging F5)** (рис. 19).

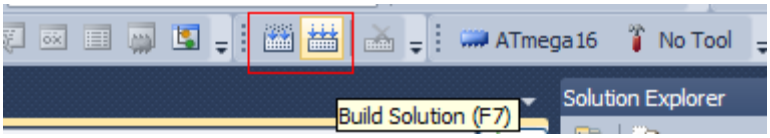

Рис. 18 – Панель компиляции

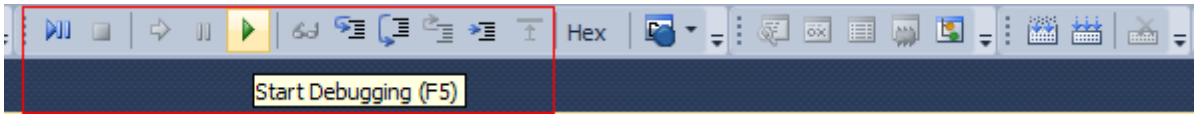

Рис. 19 – Панель отладки («дебага»)

Появится предупреждение компилятора (рис. 20). В нем указано, что не найден подключенный по USB программатор и не определена плата для программирования. Указать Simulator в качестве программатора, таким образом, на компьютере будет происходить имитация подключенной отладочной платы (рис. 21).

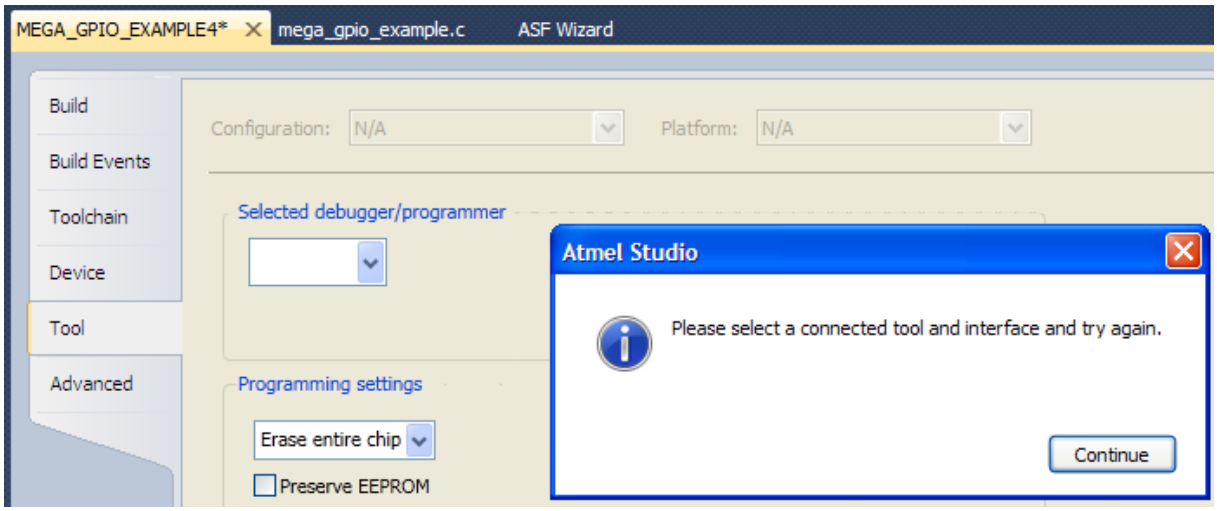

Рис. 20 – Окно выбора программатора

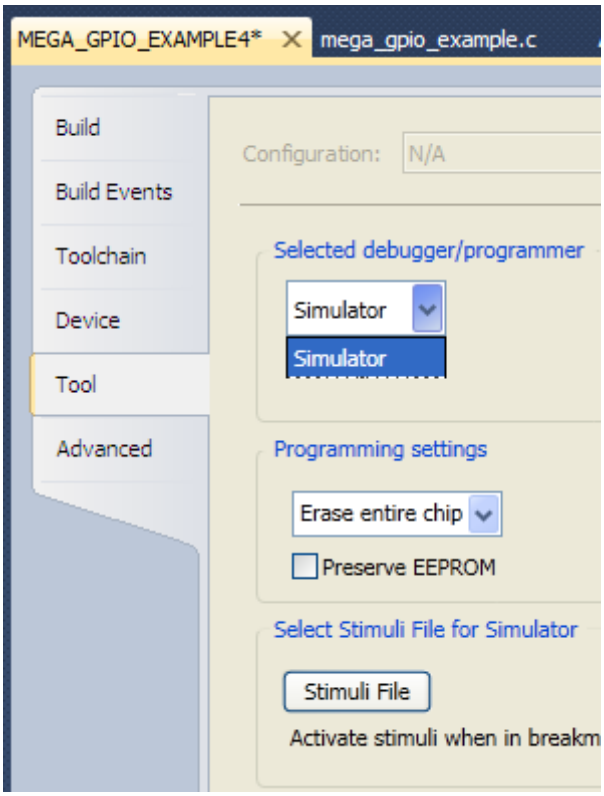

Рис. 21 – Выбор симулятора

Необходимо открыть окно периферии микроконтроллера. Для этого зайти в меню **Debug -> Windows -> I/O View** и выбрать место размещения. Обычно оно располагается в правой части экрана (рис. 22).

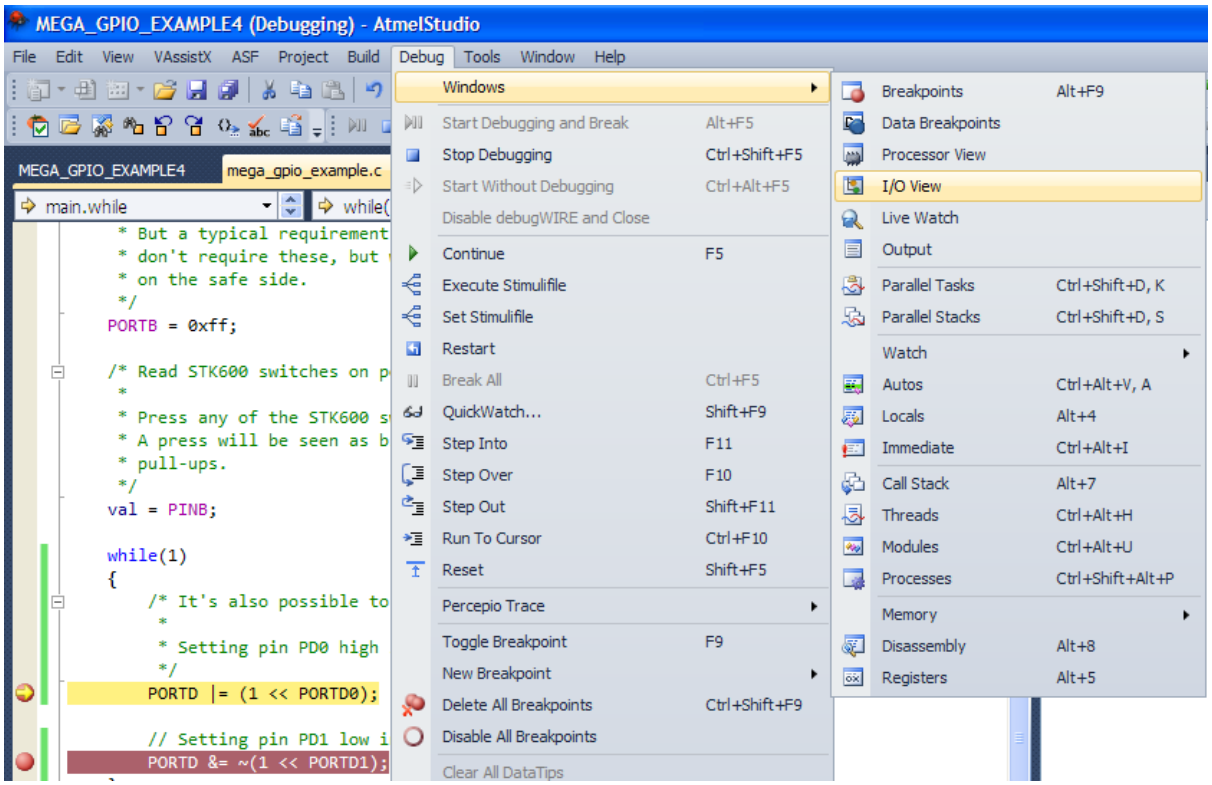

Рис. 22 – Окно выбора периферийных устройств

Выставить точки остановки («брейкпоинты») и открыть регистры состояния порта D (окно I/O View), как указано на (рис. 23). Для того чтобы установить точки останова, нужно в левой части экрана в серой области нажать мышкой. Красный круг обозначает место остановки программы, стрелочка указывает следующую команду микроконтроллера.

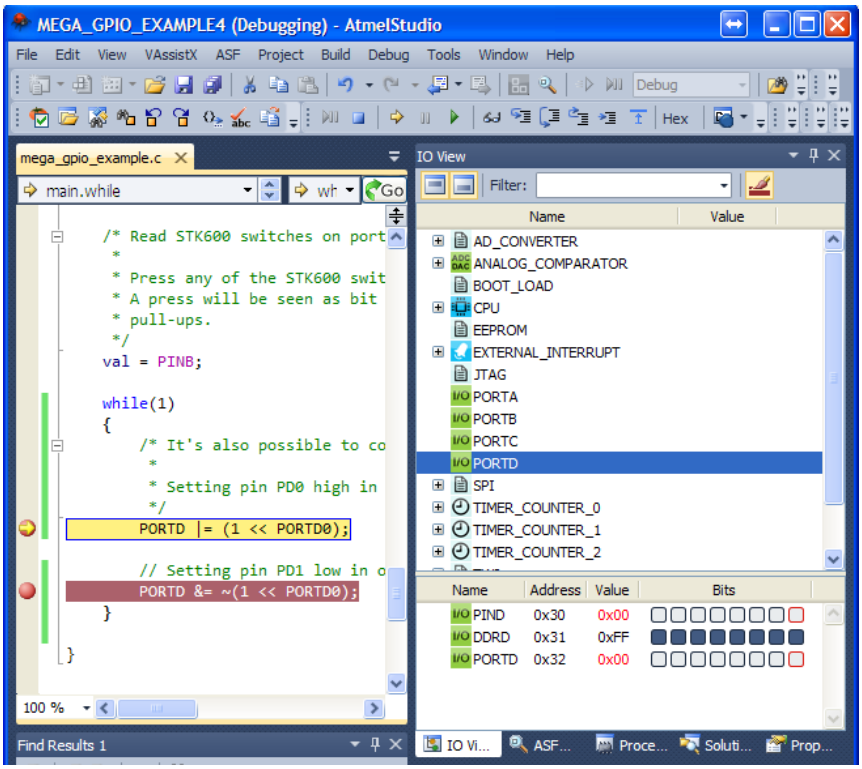

Рис. 23 – Программа с установленным брейкпоинтом

Нажать **F5** для перехода от одной точки к другой. Понаблюдать и дать комментарий, как выполняется программа работа с битами (рис. 24). Красным цветом обозначено измененное значение на порту (ножке микроконтроллера).

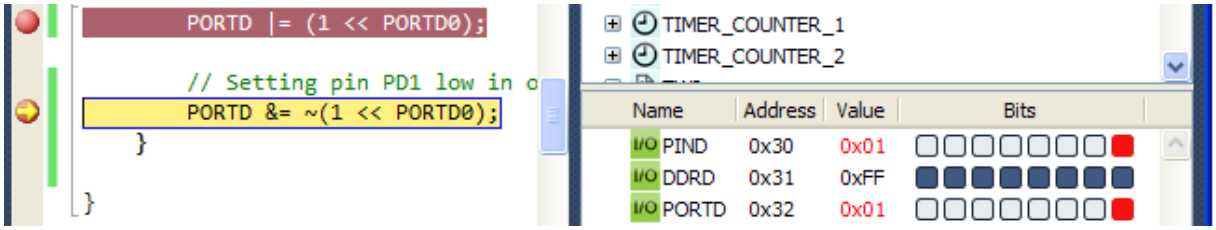

Рис. 24 – Выставление бита

Перейти на страницу дизассемблера, нажав на вкладку **Disassemble** в верхнем левом углу (рис. 25).

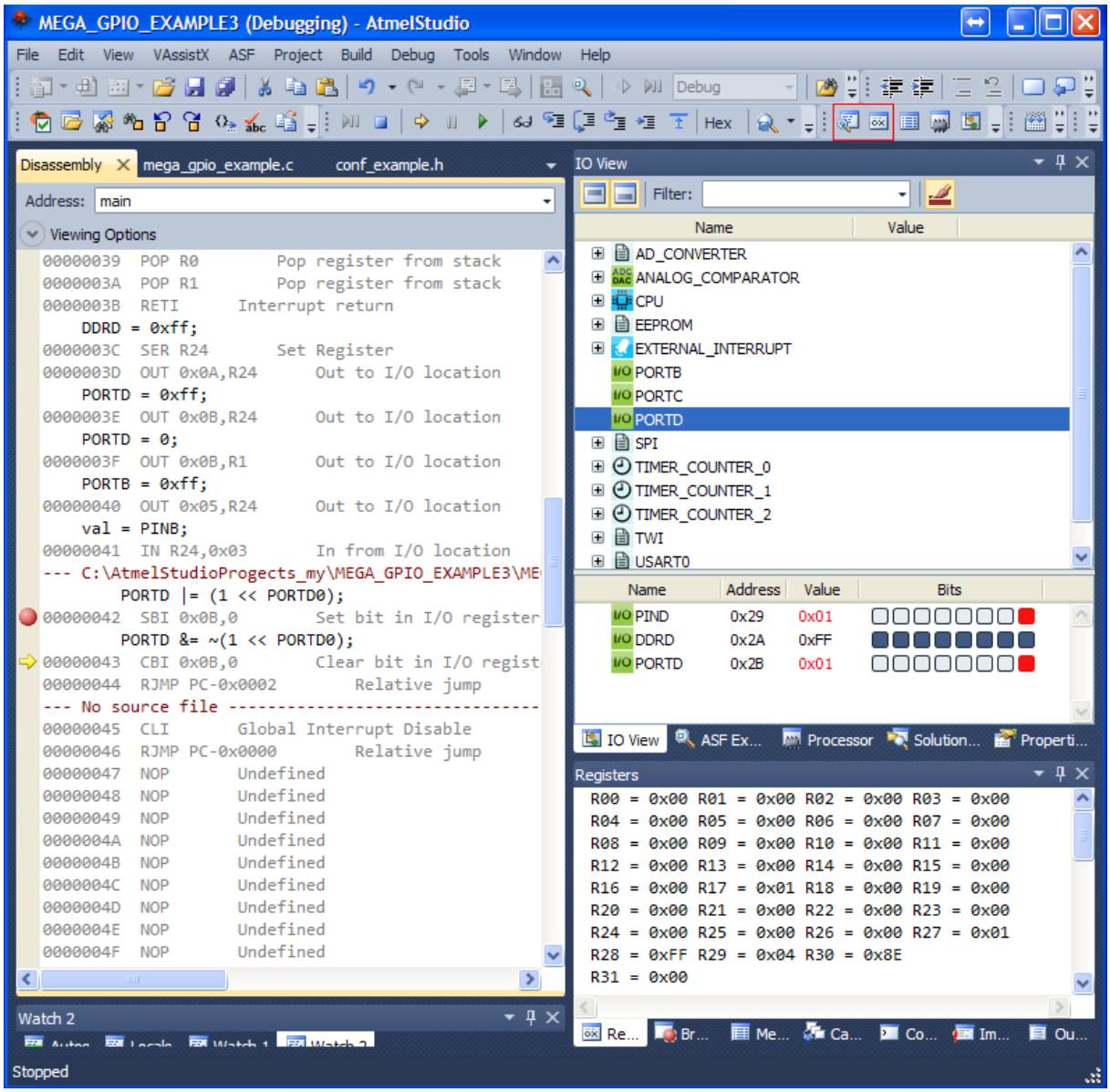

Рис. 25 – Код ассемблера

Дать комментарий, какие команды появились вместо команд:

PORTD |= (1 << PORTD0);

PORTD &= ~(1 << PORTD0);

Найти их в списке команд ассемблера. Дать комментарий, что выполняют команды NOP в конце вашей программы

Задание 2. Знакомство с регистрами порта (общее для всех вариантов)

Модифицировать программу в цикле **while(1)** так, чтобы при пошаговом режиме выполнения инвертировались значения всех светодиодов в порту **PORTD** (рис. 26). Привести код ассемблера.

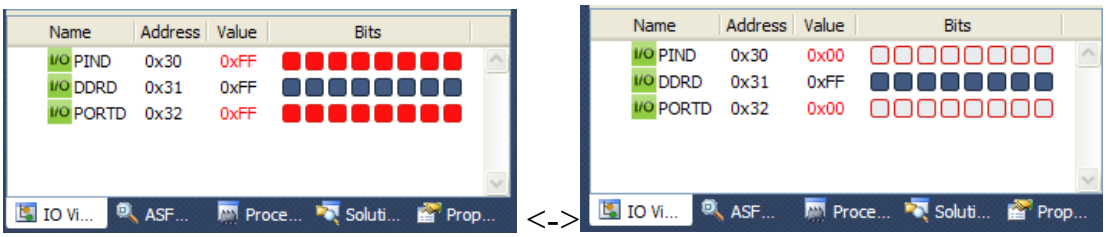

Рис. 26 – Изменение состояния на портах в задании 2

Задание 3. Создание переменных, изучение условий if(), циклов

Объявить переменную i в процедуре main,

```
int main(void)
{
	// Variable to put switch input into
	uint8_t val;
	uint8_t i;
	//
```

Модифицировать код в бесконечном цикле while(1)

```
	while(1)
	{
		PORTD |= (1 << i);
		i++;
	}
```

Установить брейкпоинт на добавленной команде (рис. 27).

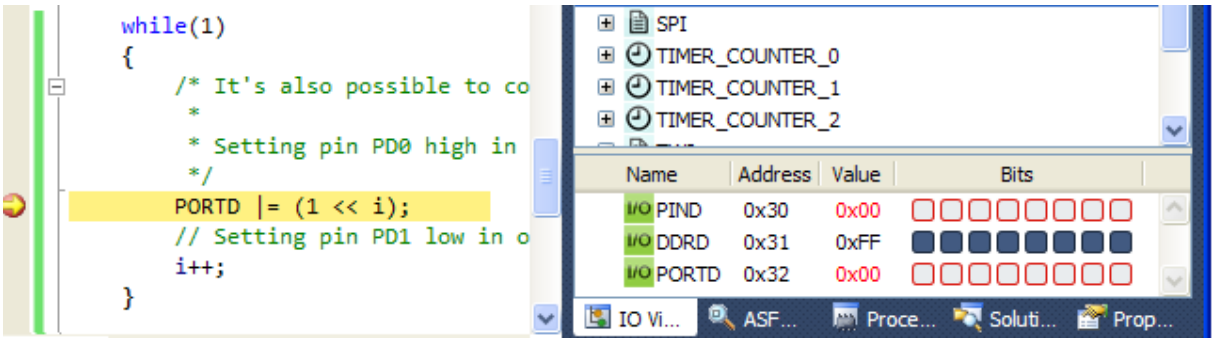

Рис. 27 – Основной бесконечный цикл программы

Нажимая кнопку **F10** (пошагово) понаблюдать новое поведение на порте (рис. 28)

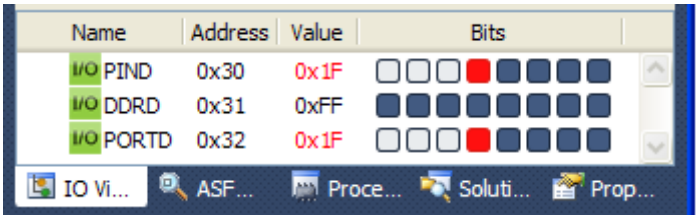

Рис. 28 – Регистры порта D

Добавить условие if(), понаблюдать за поведением на порте D.

Посмотреть код ассемблера. Найти в таблице команд ассемблера, какой код появился в месте условий if().

```
while(1)
{
	PORTD |= (1 << i);
	i++;
	if (i>8)
	{
		i=0;
		PORTD =0;
	}
}
```

Выполнить задание согласно варианту:

1. Реализовать бегущий огонек, который доходит до 8-го разряда и начинает с 1-го (…0100 0000 -> 1000 0000 -> 0000 0001…).

2. Реализовать бегущий огонек, который доходит до 8-го разряда и начинает считать в обратную сторону (…0100 0000 -> 1000 0000 -> 0100 0000 -> 0010 0000…).

3. Реализовать эффект ёлочки, который доходит до 8-го разряда и начинает считать в обратную сторону (…0111 1111 -> 1111 1111 -> 01111 1111 -> 0011 1111…).

4. Реализовать эффект ёлочки, который доходит до 8-го разряда и начинает считать с начала (…0111 1111 -> 1111 1111 -> 0000 0000 -> 0000 0001…).

5. Реализовать эффект эквалайзера (…0000 0000 –> 0001 1000 –> 0011 1100 –> 0111 1110 –> 1111 1111 –> 0111 1110 –> 0011 1100 –> 0001 1000 –> 0000 0000).

6. Реализовать инверсный эффект эквалайзера (см. п. 5, заменив 0 на 1).

7. Бегущий огонек на линиях порта C должен сменить направление, если на пяти линиях порта B установлены логические единицы.

8. Ёлочка 2 (линейка светодиодов, подключенная к линиям порта, последовательно заполняется огнями и постоянно горит, а звезда – старший бит – моргает).

9. «Бегущий огонёк» в одну сторону бежит по выводам порта B, в другую – по выводам порта D.

10. Если поступившее на выводы порта D – число чётное, то должны «мигать» выводы порта B, если нет – то порта C.

Задание 4. Изучение кнопки

Модифицировать код программы в бесконечном цикле.

```
while (1)
{
	if(PIND & (1<<PIND3))// если флаг в регистре RegX установлен
	{
		PORTD |= (1 << PORTD0);
	}
	else
	PORTD &= ~(1 << PORTD0);
}
```

Установить брейкпоинт и зайти в режим отладки (рис. 29). Нажимая кнопку **F10** (пошагово), пронаблюдать новое поведение на порте.

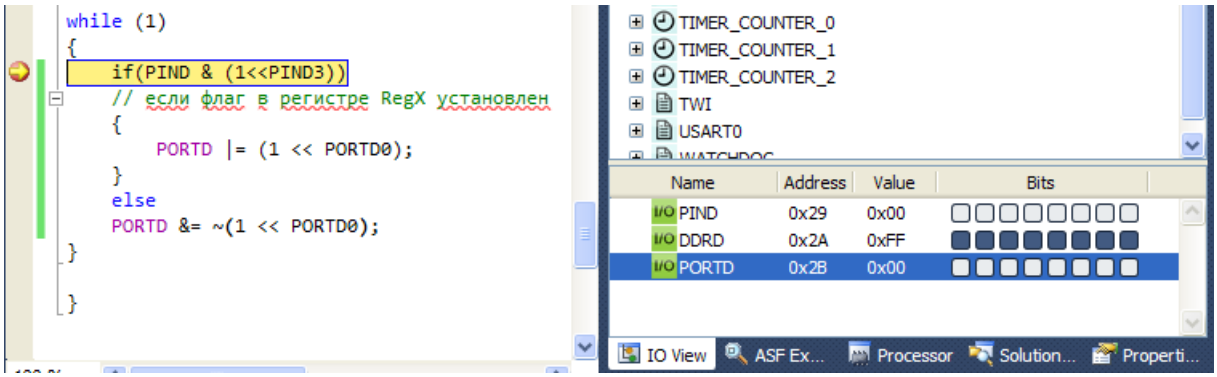

Рис. 29 – Основной цикл программы

Нажать на 3-й пин порта D, имитировав нажатие кнопки (рис. 30).

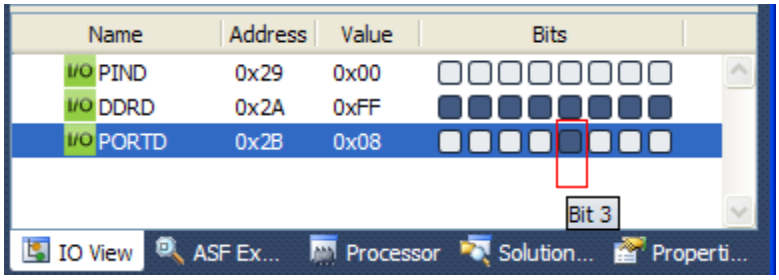

Рис. 30 – Имитация нажатия кнопки

Нажимая кнопку **F10** (пошагово) пронаблюдать новое поведение на порте (рис. 31).

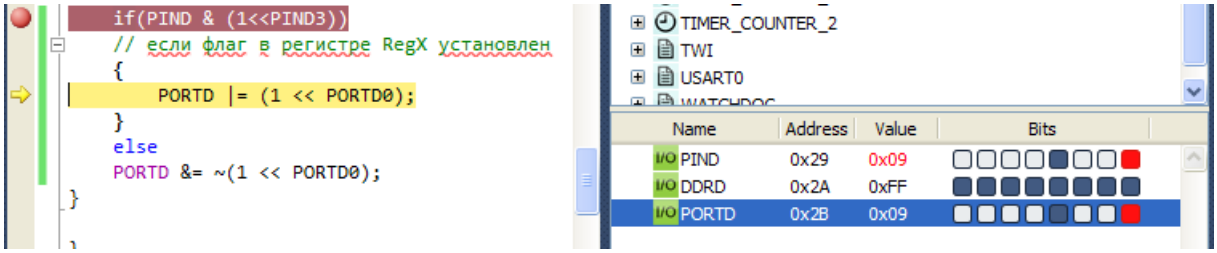

Рис. 31 – Отладка программы

Был выставлен бит на 0-й ножке.

Снова нажать на PIN 3, имитировав отпускание кнопки (рис. 32).

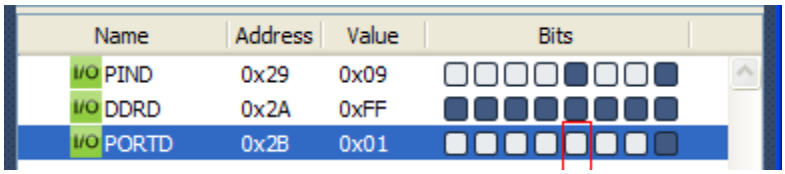

Рис. 32 – Имитация отпускания кнопки

Нажимая кнопку **F10** (пошагово), пронаблюдать новое поведение на порте (рис. 33). Светодиод на ножке 0 погас.

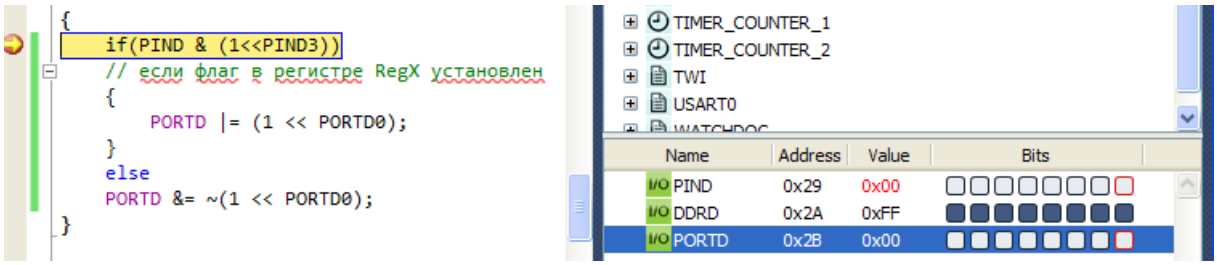

Рис. 33 – Имитация светодиодов

Модифицировать инициализацию порта D, поменяв направление на выводе PIND3. Отразить в отчете, какими C и ассемблерными командами происходит считывание данных с порта.

Задание 5. Изучение массивов, семисегментных индикаторов

Модифицировать программу, добавив после команд #include:

```
#include "compiler.h"
#include <avr/interrupt.h>
#include "conf_example.h"
```

Массив из двух переменных:

```
const char dig[2] = {0b00000111, 0b00111111};
```

Изменить бесконечный цикл while(1):

```
while (1)
{
        PORTD = dig[0];
        asm("nop");
}
```

Откомпилировать и перейти в режим отладки, понаблюдав, что происходит на порте D (рис. 34).

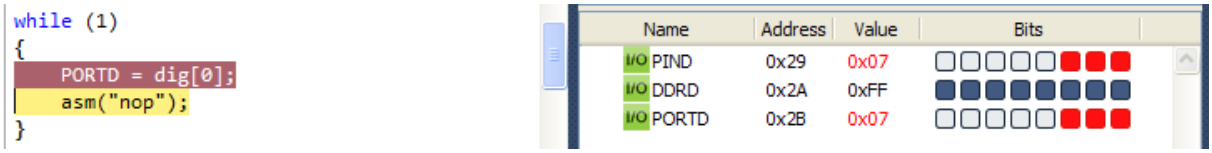

Рис. 34 – Эффект ёлочки

Открыть окно дизассемблера, найти строчку, отвечающую за вывод в порт значений из массива (рис. 35).

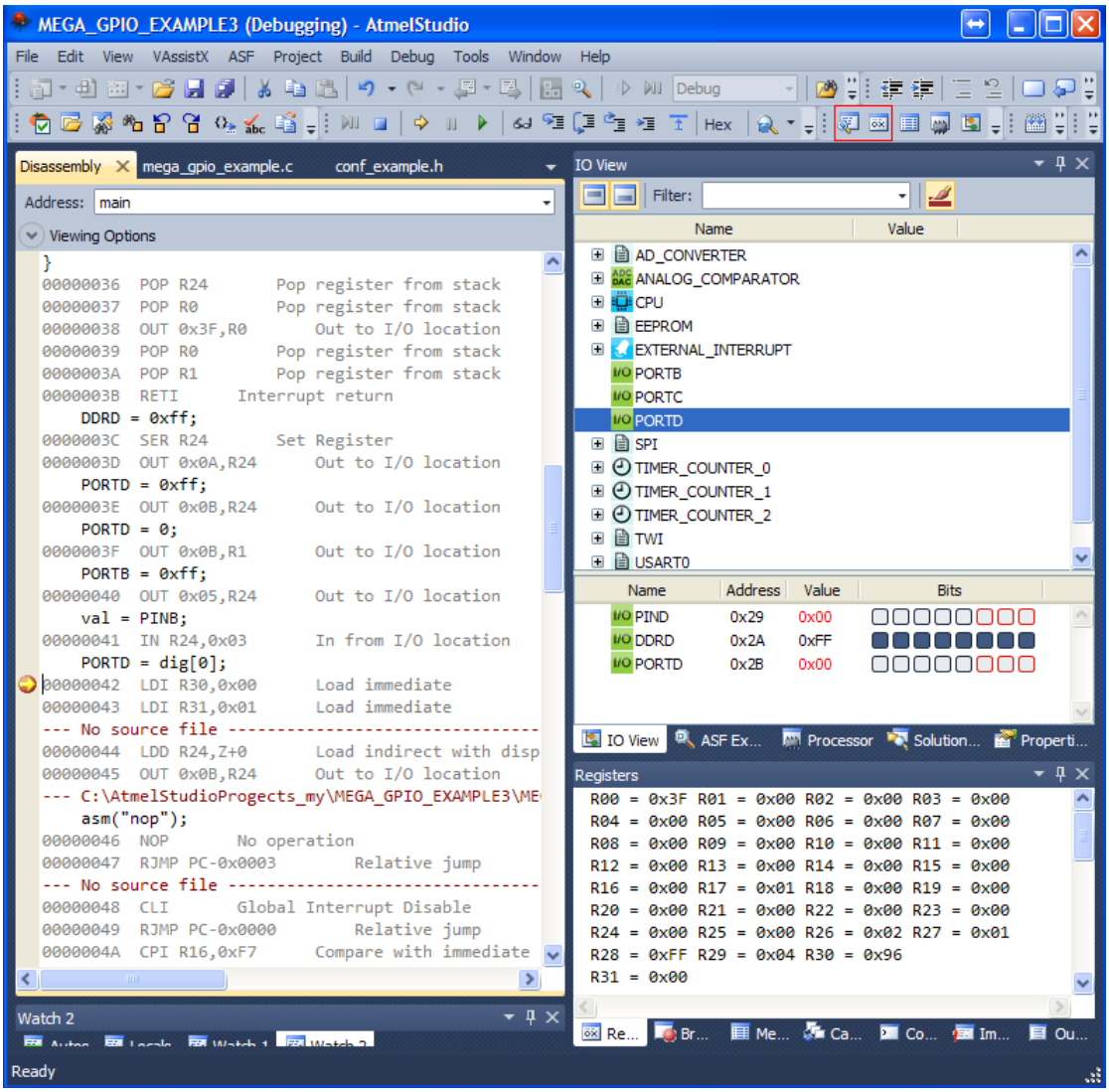

Рис. 35 – Трансляция кода программы

Задание:

– Добавить в свой вариант изменение направления эффекта при нажатии на кнопку PIN.7(PORTB) и возвращении направления при отпускании.

– Дополнительно сформировать на половине порта B бегущую единицу (…0001 –> 0010 –> 0100 –> 1000 –> 0001…), эмулирующую динамический опрос клавиатуры.

– Дополнительно на порту С, к которому подключен семисегментный индикатор, сформировать индикацию десятичного счета от 0 до 9, т. е. выводить соответствующие коды цифр индикатора.

– Привести в отчете код ассемблера итоговой программы.

Контрольные практические задания[1]

1. Преобразовать дополнительный код числа одной переменной (байт) в прямой другой переменной.

2. Преобразовать двоичный код в одной переменной (от 0 до 99) в двоично-десятичный другой переменной.

3. Просуммировать два числа в двоичном коде. Сумму, большую 255, заменить байтом единиц.

4. Вычесть два числа в двоичном коде. Разность, меньшую нуля, заменить байтом нулей.

5. Умножить два 8-разрядных числа в двоичном коде. Старший байт заменить байтом единиц.

6. Поделить два 8-разрядных числа в двоичном коде без остатка. Старший байт заменить байтом единиц.

7. Проверить число А на четность. Если четное, то выставить 4-й бит 1 в переменной Б, иначе сбросить в 0.

8. Сложить два десятичных числа (байт) в двоично-десятичном коде.

9. Реализовать суммирующий двоично-десятичный счетчик.

10. Реализовать вычитающий двоично-десятичный счетчик.

[1] Задания служат для контроля усвоения материала студентом. Не являются обязательными для выполнения. Могут быть предложены преподавателем в качестве дополнительного задания при защите лабораторной работы.

Контрольные теоретические вопросы

✓ Какие регистры обслуживают параллельный порт D микроконтроллера ATmega16? Как настроить линию порта на ввод или вывод? Как подключить к линии порта, настроенной на ввод, подтягивающий резистор?

✓ Назовите нагрузочную способность линий порта AVR.

✓ Приведите схему алгоритма разработанной по вашему варианту программы.

✓ Запишите результат выполнения арифметических операций: 245/37 и 245%37.

✓ Какими командами можно организовать задержку в одну секунду в программе для AVR на языке Си?

Содержание отчета

Отчет должен содержать листинги отлаживаемых программ на языке С и код ассемблера, комментарии по ходу выполнения пунктов работы, примеры экранных форм (скриншоты), создаваемые по ходу выполнения работы (включая промежуточные действия и финальные задания), отображающие окна регистров и памяти, а также ответы на контрольные вопросы.

2.2 Лабораторная работа № 2
«Изучение прерываний, АЦП, UART»

Цель работы

Целью лабораторной работы является имитация цифрового вольтметра на базе микроконтроллера ATmega16 с отправкой информации на персональный компьютер по интерфейсу RS-232/USB. Имитация подачи аналогового сигнала происходит с помощью регистра данных на линиях порта A (ADC0-ADC7).

Краткая теория

Аналого-цифровой преобразователь (A/D CONVERTER) служит для получения числового значения напряжения, поданного на его вход. Этот результат сохраняется в регистре данных АЦП (рис. 36). Какой из выводов («пинов») микроконтроллера будет являться входом АЦП, определяется числом, занесенным в соответствующий регистр. Величина опорного напряжения (5 В) может изменяться для повышения точности.

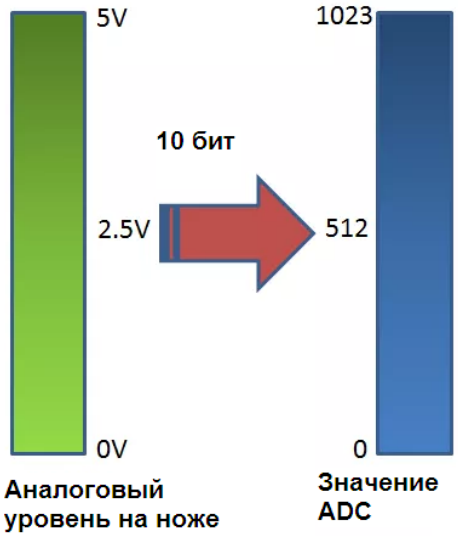

Рис. 36 – АЦП-преобразование

Управление АЦП происходит четырьмя регистрами:

– ADMUX (ADC Multiplexer Select Register) – регистр выбора мультиплексора ADC;

– ADCSRA (ADC Control and Status Register) – регистр управления и состояния ADC;

– ADC (ADCL и ADCH) – регистры данных ADC;

– SFIOR – регистр специальных функций ввода-вывода.

Универсальный асинхронный или универсальный синхронно/асинхронный приемопередатчик (Universal Synchronous/Asynchronous Receiver and Transmitter – UART или USART) – удобный и простой последовательный интерфейс для организации информационного канала обмена микроконтроллера с внешним миром.

Он совместим с протоколом стандарта RS-232, что обеспечивает возможность организации связи с персональным компьютером. Для стыковки МК и компьютера обязательно понадобится схема сопряжения уровней сигналов. Для этого существуют специальные микросхемы, например MAX232 (ADM232).

В микроконтроллере для работы с модулем USART используются 6 регистров:

– управляющий регистр UCSRA,

– управляющий регистр UCSRB,

– управляющий регистр UCSRC,

– регистры скорости передачи UBRRL и UBRRH,

– регистр данных UDR.

Система прерываний (INTERRUPTS) – одна из важнейших частей микроконтроллера.

Все микроконтроллеры AVR имеют многоуровневую систему прерываний. Прерывание прекращает нормальный ход программы для выполнения

приоритетной задачи, определяемой внутренним или внешним событием. Для каждого такого события разрабатывается отдельная процедура (либо функция), которую называют подпрограммой обработки запроса на прерывание (для краткости – подпрограммой прерывания), и размещается в памяти программ. При возникновении события, вызывающего прерывание, микроконтроллер сохраняет в стеке содержимое счетчика команд, прерывает выполнение центральным процессором текущей программы и переходит к выполнению подпрограммы обработки прерывания. После выполнения подпрограммы прерывания осуществляется восстановление предварительно сохраненного в стеке значения счетчика команд и процессор возвращается к выполнению прерванной программы. Таким образом, нет необходимости непрерывного контроля состояния флагов командами if(), while().

Программа работы

Запустить Atmel Studio 6.2 (не ниже). Выбрать **Создать новый пустой проект** (рис. 37).

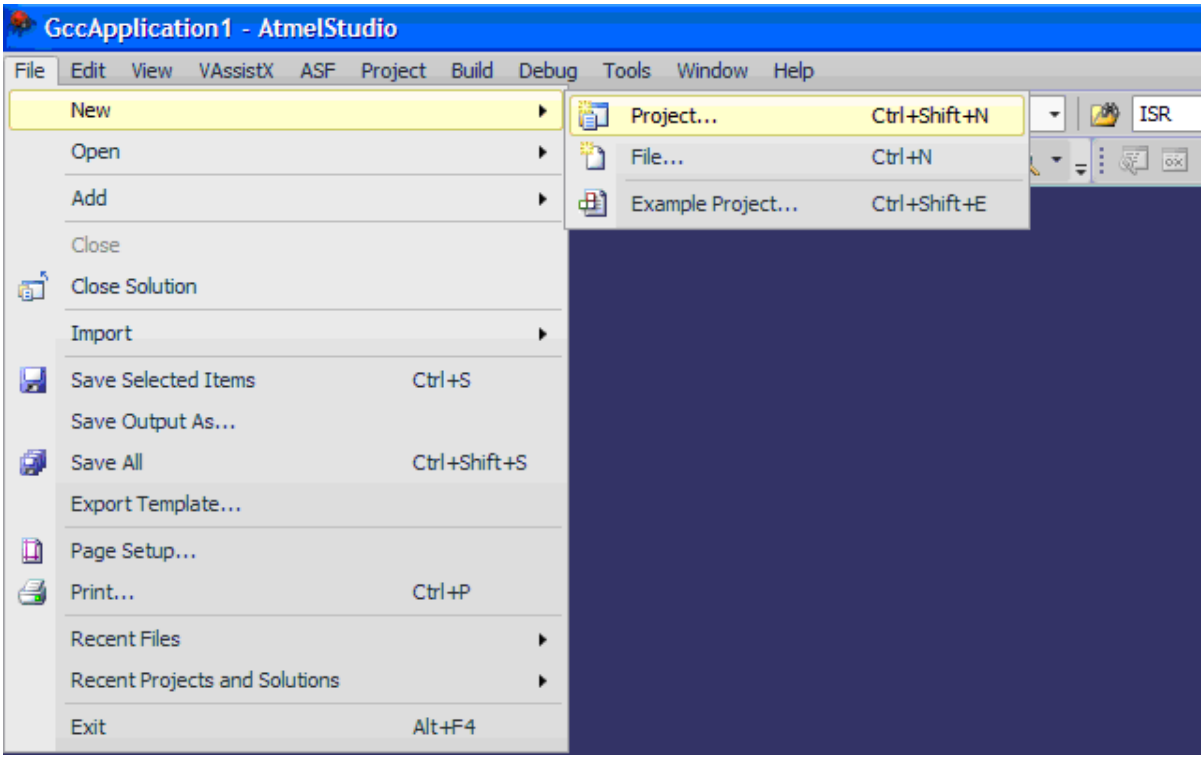

Рис. 37 – Создание нового проекта

В открывшемся окно выбрать **GCC C Executable Project** (Исполняемый проект на языке С). В нижней части указать имя проекта и путь (рис. 38). Они не должны содержать русских символов. Нажать **ОК** при готовности.

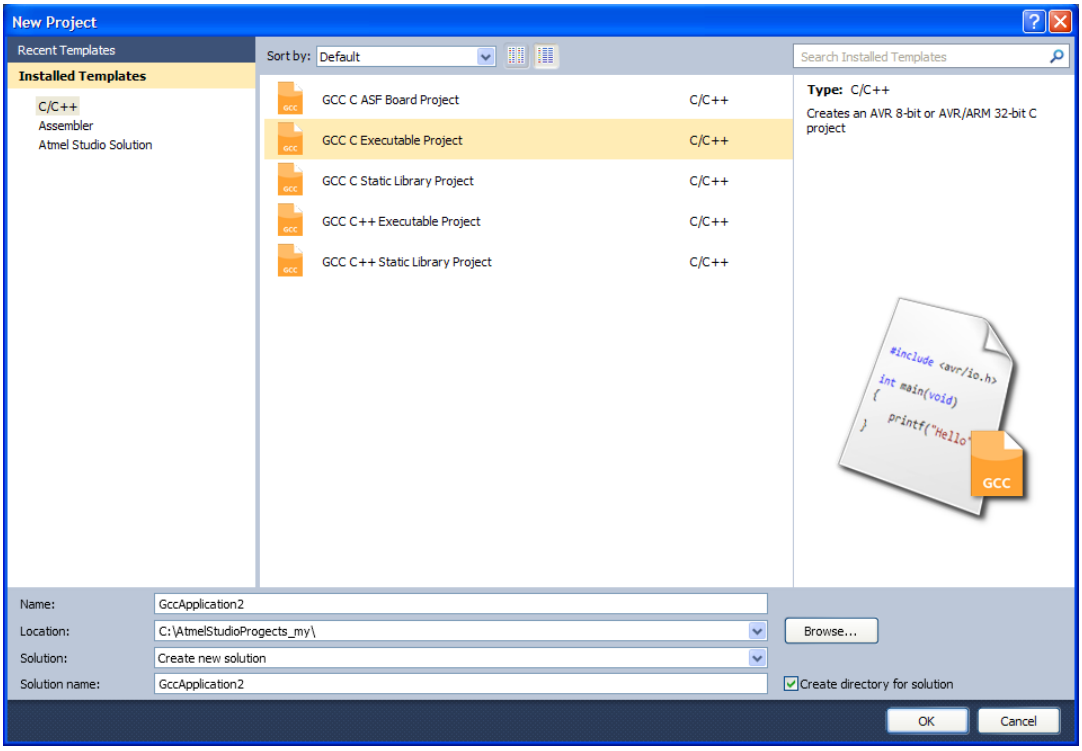

Рис. 38 – Выбор типа проекта

Выбрать в выпадающем меню «megaAVR, 8bit», контроллер ATmega16 (рис. 39).

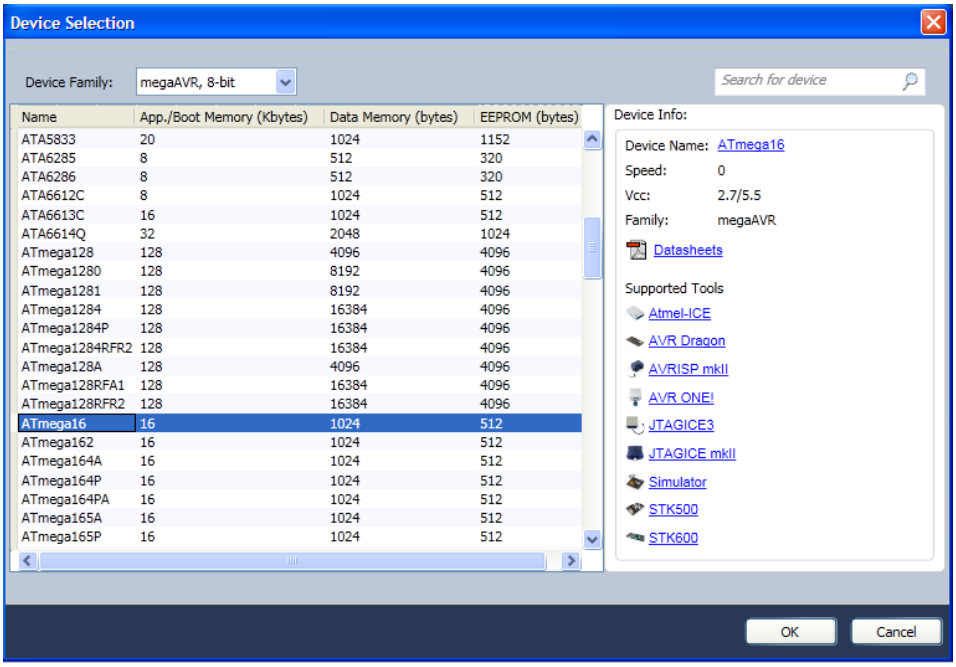

Рис. 39 – Выбор микроконтроллера

В открывшийся пустой проект скопировать приведенный текст программы.

```
#include <avr/io.h>
#include <avr/interrupt.h>

/* макроопределение, для работы с битами */
#define BIT(n)       (1<<(n))
#define ENABLE(x,n)   ((x) |= BIT(n))
#define CHECKBIT(x,n) ((x) & BIT(n))
char ADCdata;      // глобальная переменная для хранения измеренных данных

void port_init(void)
{
 PORTA = 0x00;
 DDRA = 0x00; // порт A делаем входным
 PORTB = 0x00;   // остальные порты не трогаем
 DDRB = 0x00;
 PORTC = 0x00;
 DDRC = 0x00;
 PORTD = 0x00;
 DDRD = 0x00;
}

// ADC initialize // Conversion time: 104uS
void adc_init(void)
{ // инициализация АЦП модуля
 ADCSRA = 0x00; // disable adc
 ADMUX = 0x00|(1<<ADLAR); /* будем использовать первый канал АЦП, в качестве референса потенциал поданный на вывод AREF, 8-бит разрядность АЦП и равнение налево :) */
 ACSR = 0x80; // выключаем аналоговый компаратор
 ADCSRA = 0xCD; /* включаем АЦП и запускаем одиночное преобразование, включаем прерывание по окончанию преобразования, устанавливаем частоту преобразования */
}

ISR(ADC_vect)
{
 ADCdata = ADCH; // отсылаем данные
 ENABLE(ADCSRA, ADSC); // запускаем новое преобразование
}
```

```
void init_devices(void)
{
 cli(); // на время инициализации периферии запрещаем все прерывания
 port_init();
 adc_init();
 MCUCR = 0x00; // обнуляем на всякий случай все остальные прерывания
 GICR = 0x00;
 TIMSK = 0x00; // timer interrupt sources
 sei(); // разрешаем обратно все прерывания
}

int main(void)
{
 init_devices();
 while(1) // создаем бесконечный цикл
 {

 }
}
```

Задание 1. Изучение АЦП

Установить брейкпоинт на строке ADCData = ADCH. Откомпилировать проект, нажав **F7 (Built)** и при отсутствии ошибок зайти в режим отладки **F5**.

Для отслеживания созданных переменных существуют окна **Watch1,2**. Обычно они располагаются в левой нижней рабочей области компилятора. При отсутствии, их можно добавить через верхнее выпадающее меню Window.

В окно **Watch** ввести созданную переменную ADCdata, открыть регистры, отвечающие за работу ADC – AD_CONVERTER (рис. 40). Таким образом, в окне **Watch** можно наблюдать и изменять значения ваших переменных, а в окне **I/O View** – регистров микропроцессора.

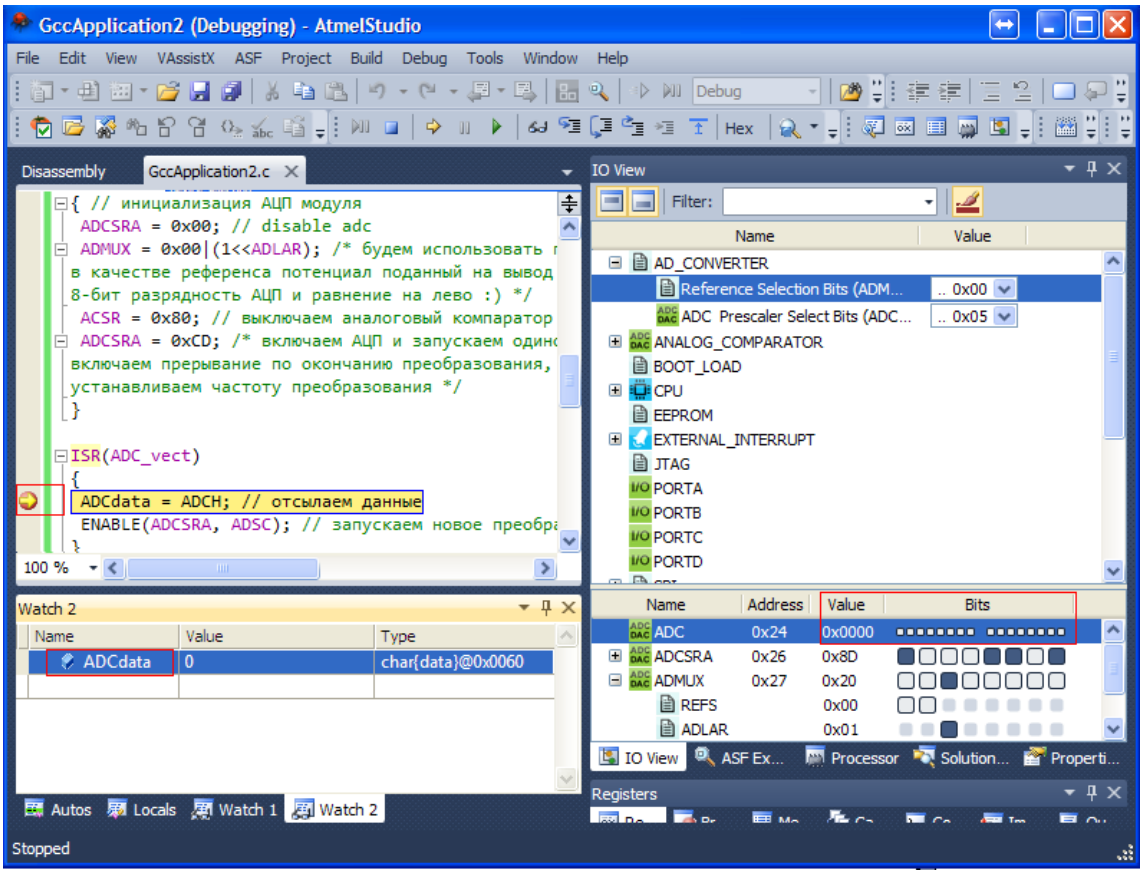

Рис. 40 – Считывание значений АЦП

Изменить значение ADC (биты) в правом окне на любое и нажать **F10**. Посмотреть значение переменной в левом нижнем окне.

Перейти в окно дизассемблера. Найти команды чтения данных из регистра ADCH, как компилятор считал данные только из 1-го регистра (ADCH) без чтения ADCL (рис. 41). Найти в дизассемблере и указать в отчете как работает макрос ENABLE(ADCSRA, ADSC).

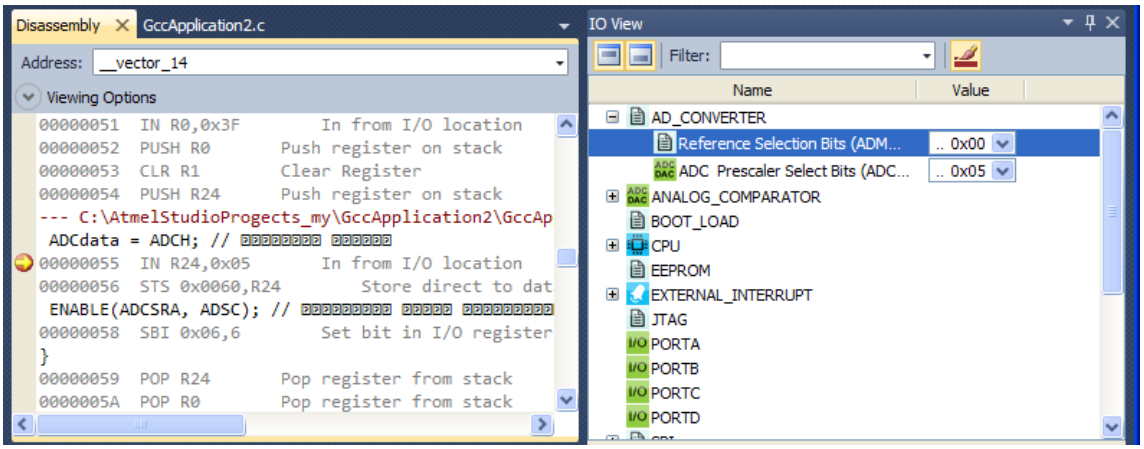

Рис. 41 – Окно дизассемблера копирования данных из АЦП

Выполнить задание согласно варианту:

1. Настроить АЦП на автоматический циклический запуск (без команды ENABLE(ADCSRA, ADSC);).
2. Настроить коэффициент усиления 10х.
3. Изменить вход АЦП на ADC3.
4. Изменить канал опорного напряжения на AVCC с внешним конденсатором на выводе AREF.
5. Изменить канал опорного напряжения на внутренний источник опорного напряжения 2.56 В с внешним конденсатором на выводе AREF.
6. Уменьшить скорость работы в 64 раза.
7. Уменьшить скорость работы в 32 раза.
8. Уменьшить скорость работы в 16 раз.
9. Установить дифференциальный вход.
10. Изменить порядок считанных данных на противоположный (результат преобразования может иметь левосторонний или правосторонний формат).

Задание 2. Изучение UART

Добавим передачу оцифрованных данных по UART (предположив, что подключен компьютер через RS-232 преобразователь).

Добавить функцию (т. к. в скобке справа usart_init указана переменная) инициализации UART в любое место между char ADCdata и процедурой main().

```
// USART initialize
// desired baud rate: 9600 actual: baud rate: 9615 (0,2%)
// char size: 8 bit parity: Disabled
void usart_init(unsigned baudrate)
{
 UCSRB = 0x00; // disable while setting baud rate
 UCSRA = 0x00;
 UCSRC = BIT(URSEL) | 0x06;
```

```
 UBRRL = baudrate;
 UCSRB = 0x08;
}
```

Модифицировать код в прерывании.

```
ISR(ADC_vect)
{
 while( !CHECKBIT(UCSRA,UDRE) ); // ждем, пока освободится буфер
 UDR = ADCH; // отсылаем данные
 ENABLE(ADCSRA, ADSC); // запускаем новое преобразование
}
```

Добавить вызов инициализации.

Переменная baudrate в функции usart_init заносится в регистр UBRRL, таким образом, число 25 в данном примере соответствует настройкам скорости. Посмотреть описание регистра UBRRL.

```
usart_init(25);
```

Установить «брейкпоинт» на строке UDR = ADCH. Откомпилировать проект, при отсутствии ошибок перейти в режим отладки **(F5)** (рис. 42). Имитируя вводом в регистр ADC оцифрованные данные, посмотреть результат в режиме отладки.

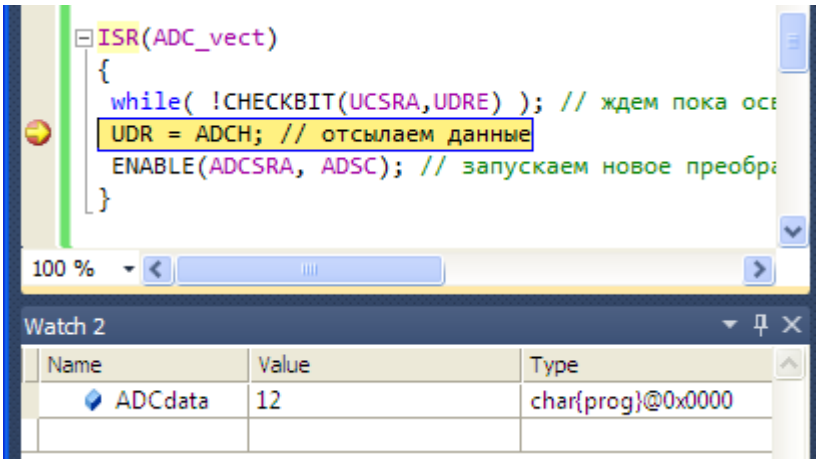

Рис. 42 – Вывод данных в параллельный порт

Какие команды ассемблера используются для передачи между регистрами (рис. 43)?

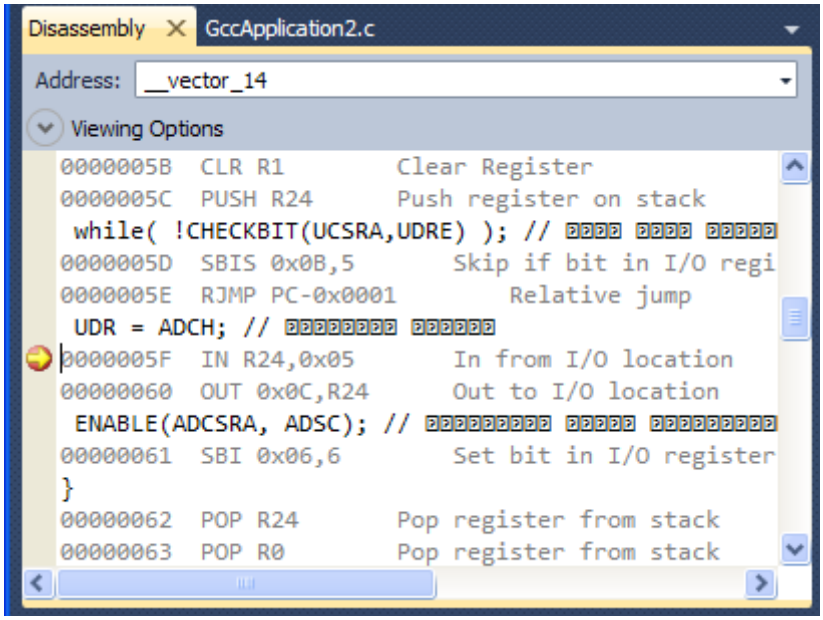

Рис. 43 – Отправка измеренных данных АЦП в порт UART

Найти и пояснить, как выполняется команда (рис. 44).

```
UCSRC = BIT(URSEL) | 0x06
```

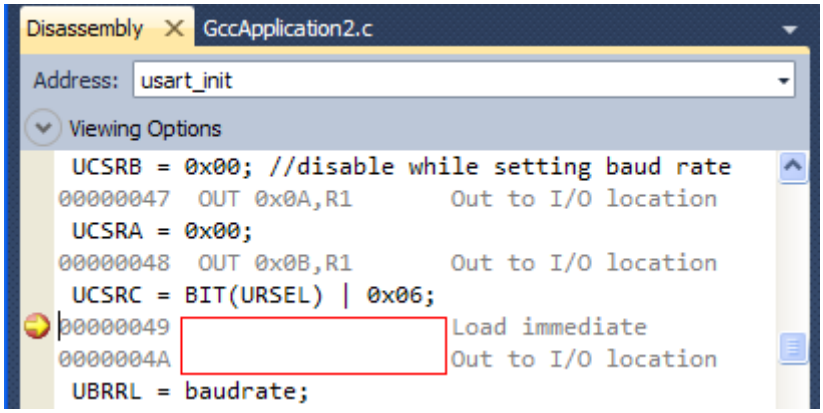

Рис. 44 – Изучение использования маски

Задание

– Изменить в своем проекте скорость передачи данных по UART в 2 раза быстрее.

– Дополнительно сделать усреднение результата оцифровки (среднее арифметическое) по четырем измерениям. Передавать усредненные данные после четырех измерений.

– Дополнительно добавить индикацию светодиодом по окончании оцифровки. Светодиод должен гореть, пока АЦП остановлено.

– Дополнительно сделать старт АЦП-преобразования и передачу данных после нажатия кнопки.

Контрольные теоретические вопросы

✓ Какой метод аналого-цифрового преобразования сигнала применяется в микроконтроллерах AVR? Назовите другие типы АЦП.

✓ Укажите диапазон тактовой частоты, рекомендуемый для работы АЦП ATmega16.

✓ Назовите возможные режимы работы АЦП.

✓ Какими ассемблерными вставками можно разрешать и запрещать глобально прерывания в программе для AVR на языке Си?

✓ Прокомментировать результат выполнения команды ADCSRA|=0x40;

Содержание отчета

Отчет должен содержать листинги отлаживаемых программ на языке С и код ассемблера, комментарии по ходу выполнения пунктов работы, примеры экранных форм (скриншоты), создаваемые по ходу выполнения работы (включая промежуточные действия и финальные задания), отображающие окна регистров и памяти, а также ответы на контрольные вопросы.

2.3 Лабораторная работа № 3 «Таймеры/счетчики, ШИМ (PWM) модуляция»

Цель работы

Целью лабораторной работы является исследование работы таймеров/счетчиков и системы прерываний.

Для выполнения работы необходимо использовать «Timer/Counter0/1/2» микроконтроллера. Представить, что выходы «OC1A» и/или «OC1B» связаны со светодиодом/светодиодами. Принять для простоты, что светимость светодиода линейно зависит от коэффициента заполнения прямоугольных импульсов, подаваемых на него.

Краткая теория

Микроконтроллеры AVR имеют в своем составе от 1 до 4 таймеров/счетчиков (TIMER/COUNTERS) с разрядностью 8 или 16 бит, которые могут работать и как таймеры от внутреннего источника тактовой частоты, и как счетчики внешних событий.

Их можно использовать для точного формирования временных интервалов, подсчета импульсов на выводах микроконтроллера, формирования последовательности импульсов. В режиме ШИМ (PWM) таймер/счетчик может представлять собой широтно-импульсный модулятор, он используется для генерирования сигнала с программируемыми частотой и скважностью.

Таймеры/счетчики способны вырабатывать запросы прерываний, переключая процессор на их обслуживание по событиям и освобождая его от необходимости периодического опроса состояния таймеров. Поскольку основное применение микроконтроллеры находят в системах реального времени, таймеры/счетчики являются одним из наиболее важных элементов.

Таймер-счетчик T0 использует два вывода микроконтроллера ATmega16. Вывод T0 (PB0) – это вход внешнего тактового сигнала. Он может применяться, например, для подсчета импульсов. Вывод OC0 (PB3) – это выход схемы сравнения таймера-счетчика. На этом выводе с помощью таймера он может формировать меандр или ШИМ-сигнал. Также он может просто менять свое состояние при срабатывании схемы сравнения (рис. 45).

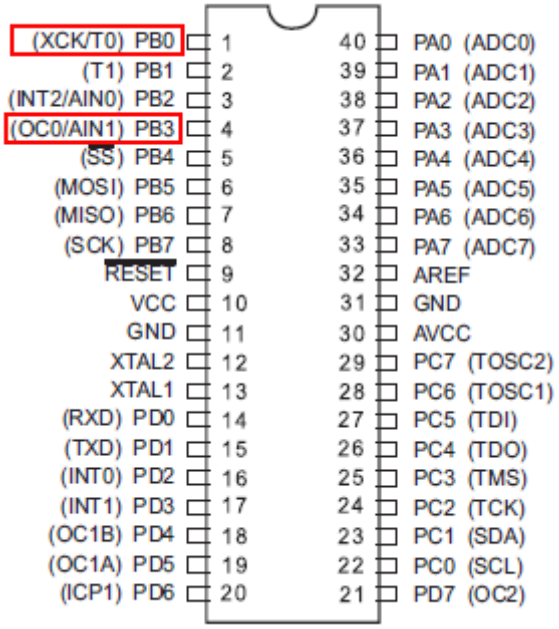

Рис. 45 – Выводы микросхемы ATmega16

Управление работой таймера/счетчика 1 осуществляется с помощью регистров, описанных в таблице 1. Настройка таймеров 0/2 происходит в своих регистрах.

Таблица 1 – Описание регистров управления таймера/счетчика1

Регистр	Краткое описание
TCNT1	Timer/Counter1 – регистр, содержащий текущее значение таймера/ счетчика 1
TCCR1A	Timer/Counter1 Control Register A – регистр задания режимов таймера/счетчика 1
TCCR1B	Timer/Counter1 Control Register B – регистр задания режимов таймера/счетчика 1

Окончание табл. 1

Регистр	Краткое описание
OCR1A	Timer/Counter Output1 Compare Register A – выходной регистр компаратора A
OCR1B	Timer/Counter Output1 Compare Register B – выходной регистр компаратора B
ICR1	Timer/Counter1 Input Capture Register1 – входной регистр защелки 1-го таймера
TIMSK	Timer Interrupt Mask Register – регистр маски прерываний
TIFR	Timer Interrupt Flag Register – регистр флагов прерываний

Программа работы

Запустить Atmel Studio. Выбрать **File – New – Project…** – новый пустой проект (рис. 46).

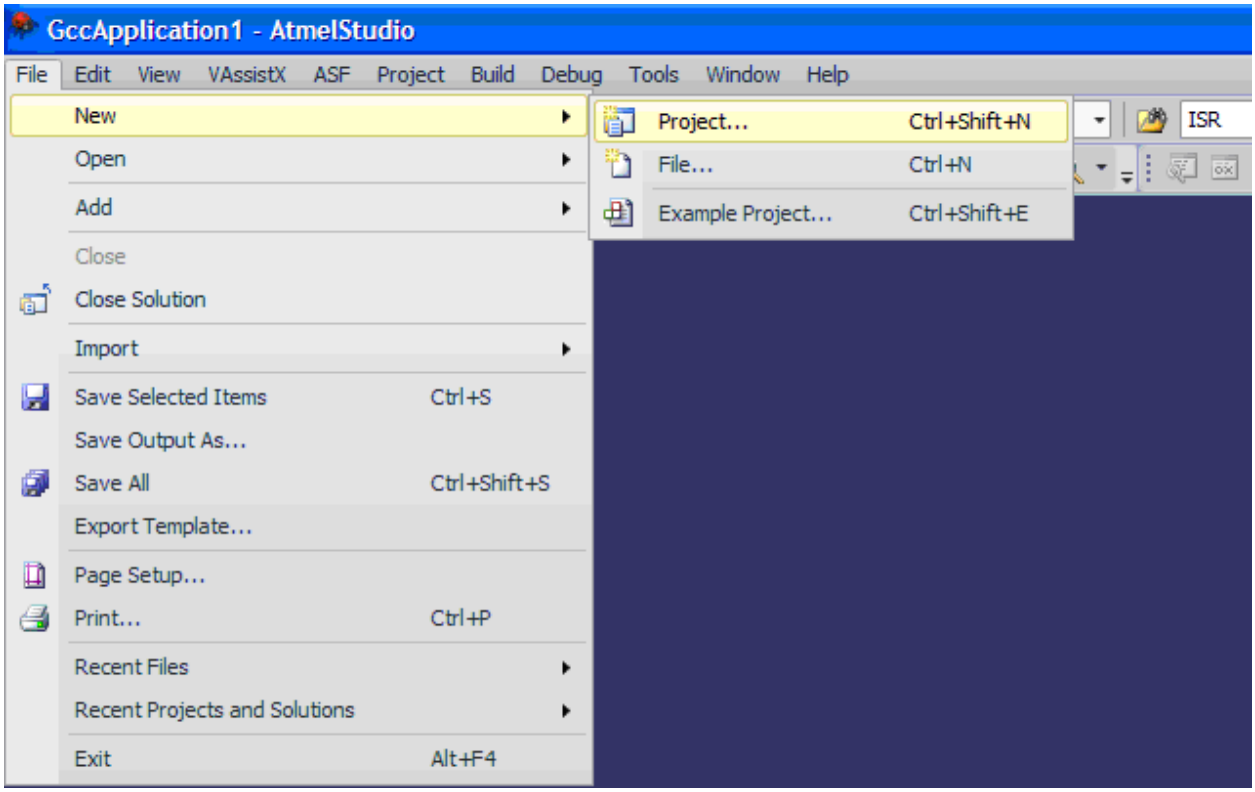

Рис. 46 – Окно создания проекта

В открывшемся окно выбрать **GCC C Executable Project** (Исполняемый проект на языке C). В нижней части указать имя проекта и путь (рис. 47). Они не должны содержать русских символов. Нажать **OK** при готовности.

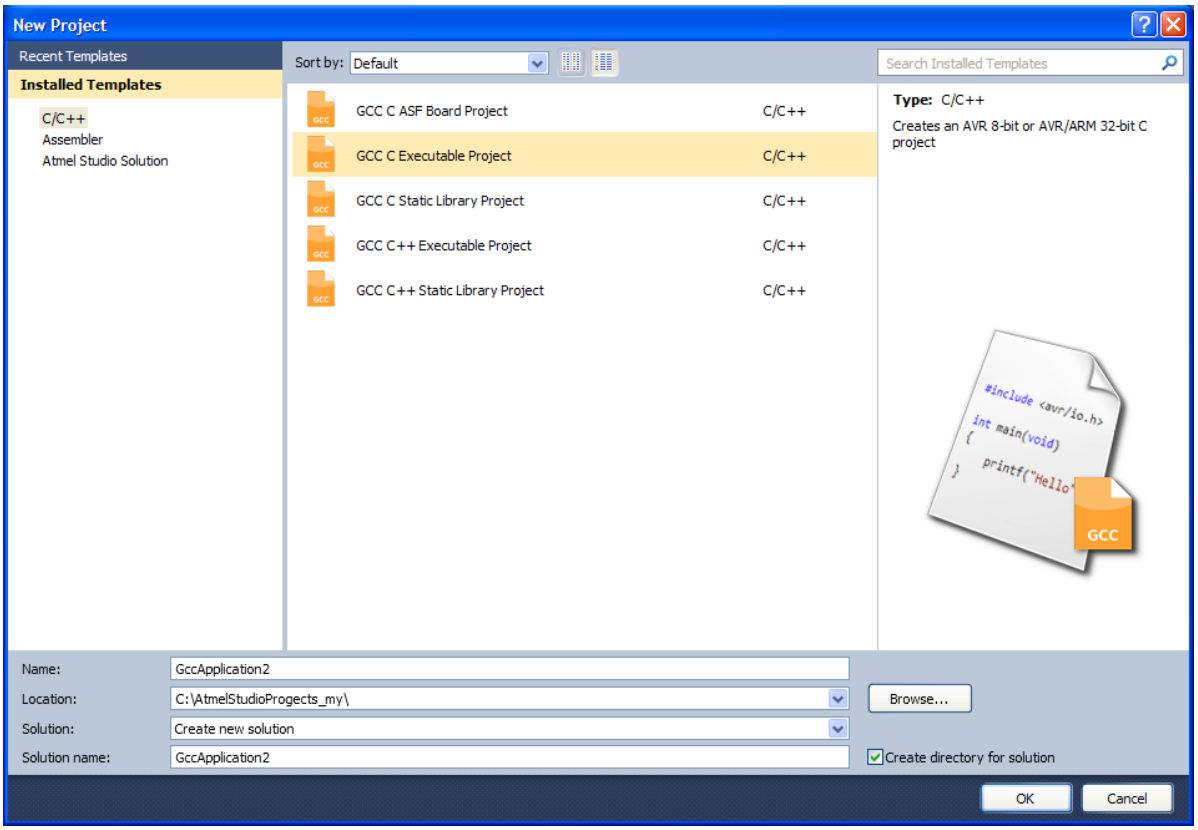

Рис. 47 – Выбор типа проекта

В следующем окне в выпадающем меню выбрать «megaAVR, 8bit», контроллер ATmega16.

В открывшийся пустой проект скопировать приведенный текст программы.

```
#include <avr/io.h>
#include <avr/interrupt.h>

void init_timer1(void) // Инициализация таймера/счетчика21
{
    DDRB = ( 1 << PB0 ); // настраиваем PB0 на выход
    TCCR1B = (0<<CS12)|(0<<CS11)|(1<<CS10); // настраиваем делитель
    TIMSK |= (1<<TOIE1); // разрешаем прерывание по переполнению
таймера
    TCNT1 = 64456;       // выставляем начальное значение TCNT1
}

ISR(TIMER2_OVF_vect)
{
  if ((PORTB & 3) == 1)
```

```
  {
    PORTB &= (0xFF << 2);       // Отключение выводов PB0, PB1
    PORTB |= 2;                 // Включение PB1
  }
  else
  {
    PORTB &= (0xFF << 2);    // Отключение выводов PB0, PB1
    PORTB |= 1;              // Включение PB0
  }
}

ISR( TIMER1_OVF_vect )
{
    TCNT1 = 64456; // выставляем начальное значение TCNT1
    if( PINB & ( 1 << PB0 ) ) {
       PORTB &= ~( 1 << PB0 );
    }
    else {
       PORTB |= ( 1 << PB0 );
    }
}

int main()
{
    init_timer1();
    sei();              // выставляем бит общего разрешения прерываний
    while(1)
    {
       asm("nop");
    };           // вечный цикл

return 0;
}
```

Задание 1. Изучение прерывания по переполнению

Установить «брейкпоинт» на строке ISR (рис. 48). Откомпилировать проект, нажав **F7 (Built)**, и при отсутствии ошибок зайти в режим отладки **F5**.

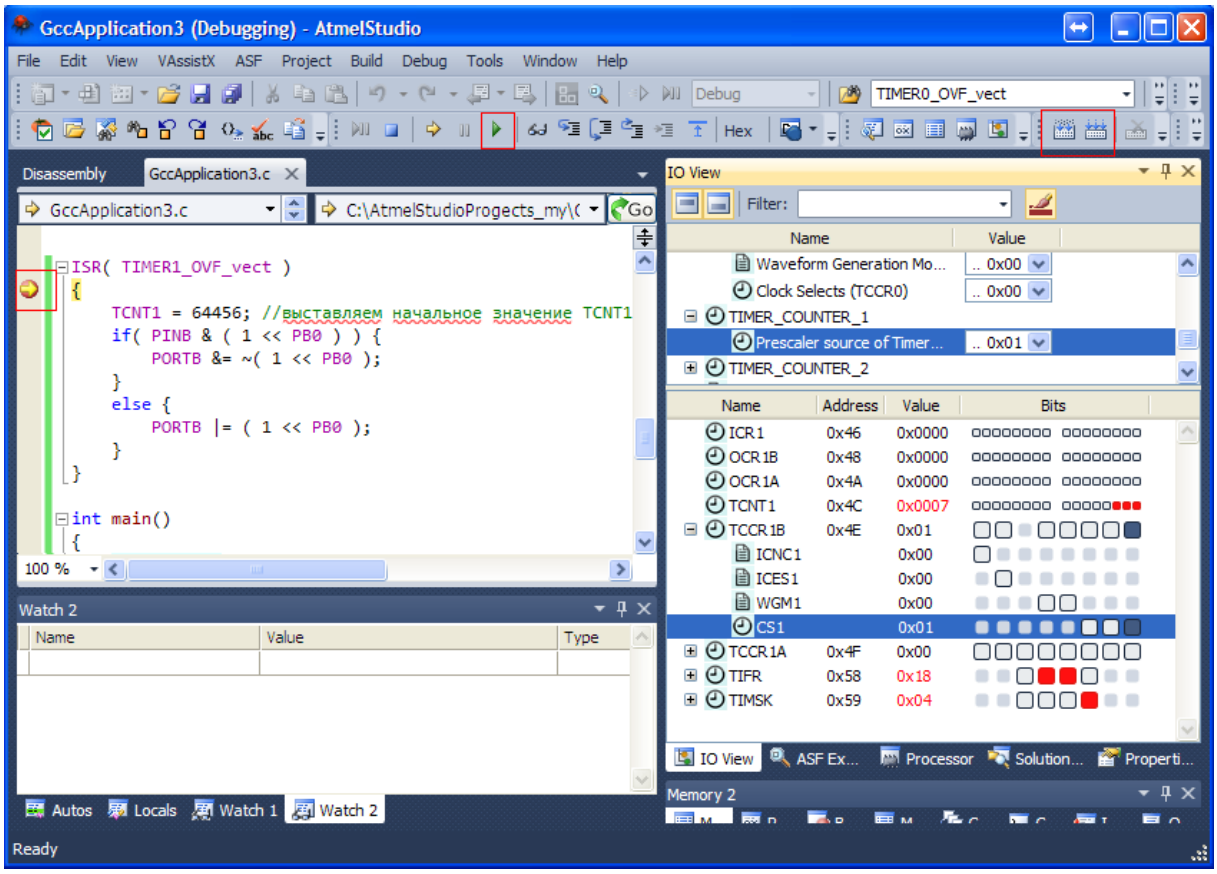

Рис. 48 – Настройка таймера/счетчика 1

Посмотреть состояние регистров (рис. 49). В каком режиме работает таймер, до какого значения происходит счет? Что происходит с выводом PB0?

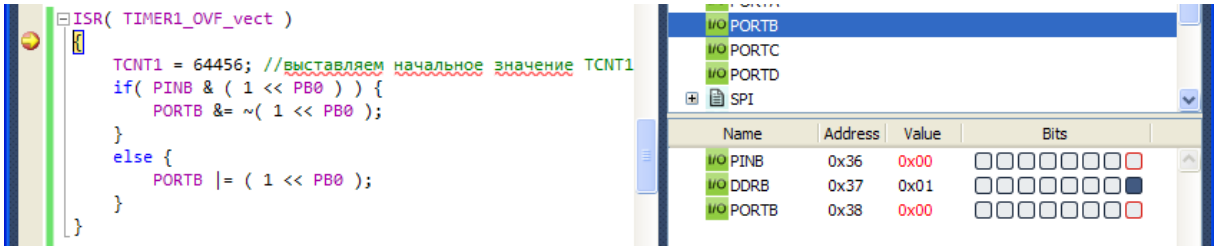

Рис. 49 – Состояние порта B

Установить «брейкпоинт» на пустой команде asm («nop»). Нажимая **F10** (пошагово), посмотреть, как происходит счет в регистре TCNT1 (рис. 50). Когда происходит выставление флагов таймера 1?

Рис. 50 – Работа таймера/счетчика

Задание 2. Изучение двух работающих таймеров

Добавить процедуры инициализации таймера 0, 1, 2 перед процедурой main().

```
 void init_timer0(void)   // Инициализация таймера/счетчика0
 {
  OCR0 = 255;          // Содержимое регистра сравнения

  // Задаем режим работы таймера
  TCCR0 = (0 << WGM01) | (0 << COM00) | (0 << CS02) | (1 << CS00);
 }

 void init_timer1(void)  // Инициализация таймера/счетчика1
 {
  DDRB = ( 1 << PB0 );  // настраиваем PB0 на выход
  TCCR1B = (0<<CS12)|(0<<CS11)|(1<<CS10); // настраиваем делитель
  TIMSK |= (1<<TOIE1); // разрешаем прерывание по переполнению таймера
  TCNT1 = 64456;       // выставляем начальное значение TCNT1
 }

 void init_timer2(void)  // Инициализация таймера/счетчика2
 {
  OCR2 = 255;
  TCCR2 = (0 << WGM21) | (0 << CS22) | (0 << CS21) | (1 << CS20);
 //  TIMSK |= (1 << OCIE2); // Устанавливаем для него прерывание совпадения
 }

 ISR(TIMER2_OVF_vect)
 {
  if ((PORTB & 3) == 1)
  {
   PORTB &= (0xFF << 2);       // Отключение выводов PB0, PB1
   PORTB |= 2;                 // Включение PB1
  }
```

```
  else
  {
   PORTB &= (0xFF << 2);    // Отключение выводов PB0, PB1
   PORTB |= 1;              // Включение PB0
  }
 }
```

В основную программу добавить вызов процедур инициализации таймеров 0,2.

```
init_timer2();
init_timer0();
```

Установить брейкпоинт на пустой команде asm («nop»). Нажимая кнопку **F10** (пошагово), посмотреть, что происходит с таймерами 0, 1, 2 в регистрах TCNT0, 1, 2. Какое вызывается прерывание?

Добавить в проект второе прерывание по таймеру 0.

```
void init_timer0(void)     // Инициализация таймера/счетчика 0
{
        OCR0 = 255;          // Содержимое регистра сравнения

        // Задаем режим работы таймера
        TCCR0 = (1 << WGM01) | (0 << COM00) | (0 << CS02) | (1 << CS00);
        TIMSK |= (1 << TOIE0);  // Устанавливаем для него прерывание по
переполнению

}

ISR(TIMER0_OVF_vect)
{
        asm("nop");
}
```

Зайти в режим отладки, посмотреть, как срабатывают два прерывания. В каких режимах работают таймеры? При каких условиях? До какого значения происходит счет? Какие флаги выставляются?

Задание 3. Изучение режима сравнения – ШИМ

Модифицировать процедуру инициализации таймера 0 и, добавив дополнительное прерывание по сравнению, установить OCR0 = 127.

```
 void init_timer0(void)   // Инициализация таймера/счетчика 0
 {
   OCR0 = 127;//255;          // Содержимое регистра сравнения
   // Задаем режим работы таймера
   TCCR0 = (1 << WGM01) | (0 << COM00) | (0 << CS02) | (1 << CS00);
 //TIMSK |= (1 << TOIE0);  // Устанавливаем для него прерывание по переполнению
   TIMSK |= (1 << OCIE0);  // Устанавливаем для него прерывание по сравнению
 }

 ISR(TIMER0_OVF_vect)
 {
   asm("nop");
 }
 ISR(TIMER0_COMP_vect)
 {
   asm("nop");
 }
```

Зайти в режим отладки **(F5)**. Развернуть регистры состояния таймера 0. В пошаговом режиме посмотреть, что происходит с флагом OCF0 (рис. 51).

AVR studio 6.2 имеет программные ошибки при работе с таймерами и прерываниями, которые, возможно, были исправлены в новой версии. При пошаговом режиме флаг сравнения выставляется ПРАВИЛЬНО, однако переход на вектор прерывания не происходит. При нажатии на **F5** (запуск) возможно зависание программы. Выход из режима отладки и вход в него снова помогут избежать зависаний. На официальном сайте разработчика эмулятора указано, что данная ошибка известна.

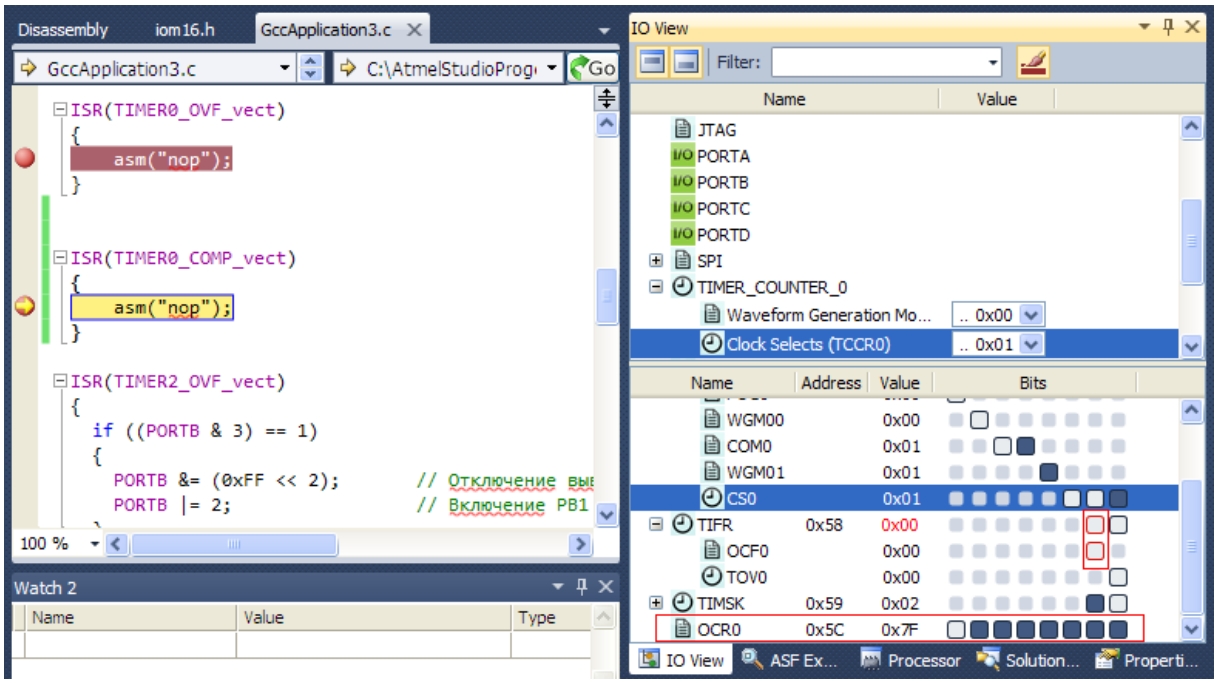

Рис. 51 – Настройка таймера 0 в режиме сравнения

Модифицировать вектор прерывания таймера 0.

```
ISR(TIMER0_COMP_vect)
{
        OCR0 = OCR0 + 1;
        asm("nop");
}
```

Зайти в режим отладки. В каком режиме работает таймер? При каких условиях? До какого значения происходит счет? Какие флаги выставляются? Посмотреть, когда выставляется флаг OCF0. Данная запись позволяет изменять уровень сравнения для формирования модулированного ШИМ-сигнала. Например, в данной записи происходит постоянное прибавление единицы, что приводит к формированию «пилообразной» развертки. Если требуется сформировать модулированный синусоидальной разверткой ШИМ-сигнал, нужно OCR0 изменять по синусоидальному закону sin (2 × 3.14 × F).

Задание 4. Изучение счетчика импульсов

Изменить настройки таймера 1 в проекте.

```
void init_timer0(void) // Инициализация таймера/счетчика 0
{
 OCR0 = 127;//255;          // Содержимое регистра сравнения
 DDRB = 1;
 // Задаем режим работы таймера
 TCCR0 = (1 << WGM01) | (1 << COM00) | (1 << CS02) | (1 << CS01) | (1
<< CS00);
 TIMSK |= (1 << TOIE0); // Устанавливаем для него прерывание по пере-
полнению
//  TIMSK |= (1 << OCIE0); // Устанавливаем для него прерывание по
сравнению
}

ISR(TIMER0_OVF_vect)
{
 asm("nop");
}
```

Установить брейкпоинт на пустой команде asm («nop»). Нажимая кнопку **F10** (пошагово), посмотреть, что происходит с таймерам 0 (TCNT0). Какое вызывается прерывание? Посмотреть настройки таймера 0 (рис. 52).

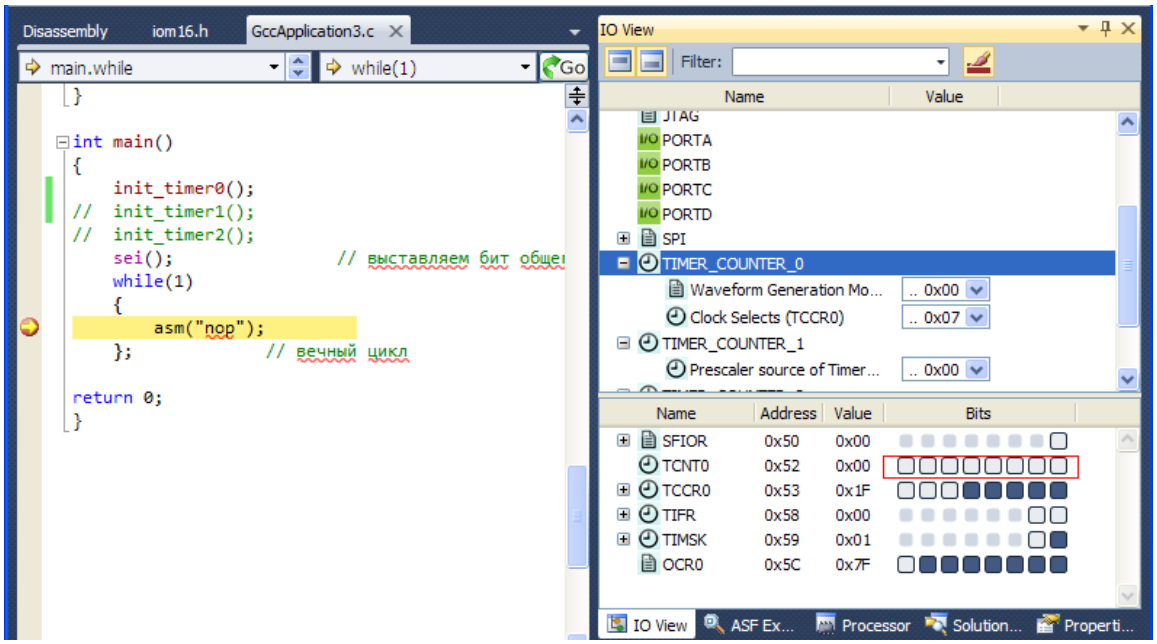

Рис. 52 – Регистры таймера 0

В режиме отладки перейти на регистр порта B. Проимитировать поступление импульса на ножку микроконтроллера. Для этого установить PINB.0 в единичное состояние (рис. 53). Нажать на **F10** (пошагово).

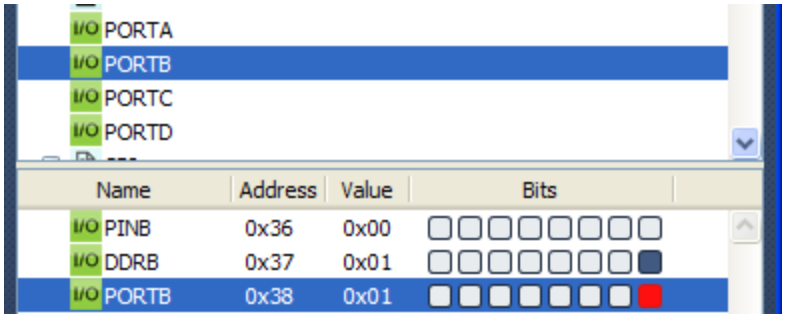

Рис. 53 – Имитация нажатий кнопки

Перейти на регистр состояния таймера 0 (рис. 54). Как изменилось значение TCNT0?

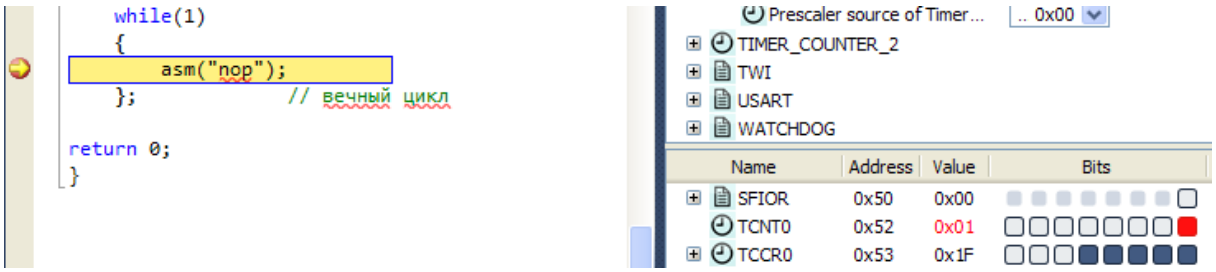

Рис. 54 – Работа таймера в режиме счета

Сбросить бит порта B. Посмотреть, как изменится значение TCNT0.

Выполнить задание согласно варианту:

1. На выводе OC0 (PB3) имитировать изменение яркости светодиода (… – плавное нарастание – плавный спад – …), «новогодняя гирлянда», для одного цвета. Для этого сформировать на ножке ШИМ-сигнал FAST PWM, модулированного треугольной разверткой.

2. На выводе OC0 (PB3) имитировать изменение яркости светодиода (… – плавное нарастание – плавный спад – …), «новогодняя гирлянда», для одного цвета. Для этого сформировать на ножке ШИМ-сигнал PWM Phase correct, модулированного синусоидальной разверткой.

3. На выводе OC0 (PB3) имитировать изменение яркости светодиода (… –плавное нарастание – мигнуть 3 раза – плавное нарастание – …), «новогодняя гирлянда», для одного цвета. Для этого сформировать на ножке ШИМ-сигнал FAST PWM, модулированного нарастающей пилообразной разверткой.

4. На выводе OC0 (PB3) имитировать изменение яркости светодиода (… – лавное гашение – мигнуть 3 раза – плавное гашение – …), «новогодняя гирлянда», для одного цвета. Для этого сформировать на ножке ШИМ-сигнал FAST PWM, модулированного спадающей пилообразной разверткой.

5. Реализовать подсчет числа импульсов (оборотов двигателя/компьютерного вентилятора), поступающих на ножку процессора с помощью Timer/Counter0 и Timer/Counter1 за 1 секунду. Один таймер ведет счет импульсов, другой отсчитывает 1 секунду.

6. Реализовать измерение длительности импульса (режим захвата таймера), поданного на ножку контроллера с помощью Timer/Counter0 и Timer/Counter1. Один таймер ведет фиксацию интервала, другой ведет измерение длительности.

7. На выводе OC0 (PB3) имитировать изменение яркости светодиода от числа нажатий на кнопку (управление режимами светодиодного фонарика). Одно нажатие – минимальная яркость, второе нажатие для средней яркости, третье нажатие – максимальная яркость, после четвертого нажатия происходит повтор. Для этого вывести ШИМ-сигнал на вывод OC0 и изменять скважность от числа нажатий. Максимальное значение ШИМ соответствует максимальной яркости (1), минимальное – минимальной яркости (0), среднее соответствует половинной скважности/яркости ($\gamma = 0.5$).

8. На выводе OC0 (PB3) имитировать изменение яркости светодиода от величины измеренных с помощью АЦП-данных (управление яркостью светодиодного фонарика). Для этого запустить АЦП в любом режиме. Максимальное значение АЦП соответствует максимальной яркости (1), минимальное – минимальной яркости (0), среднее соответствует половинной скважности/яркости ($\gamma = 0.5$).

9. Сформировать измерение данных АЦП в строго заданные таймером промежутки времени для стандартной частоты записи CD-дисков/аудио файлов 44.1 кГц (микрофонная запись голоса CD-диска). Для этого запустить таймер с частотой 44.1 кГц, в котором производить считывание данных с АЦП и запуск АЦП с максимальной скоростью так, чтобы к следующему входу в прерывание по таймеру АЦП закончило преобразование.

10. Сформировать измерение данных АЦП в строго заданные таймером промежутки времени для стандартной частоты записи DVD-аудио-дисков/аудиофайлов 96 кГц (микрофонная запись голоса DVD-диска). Для этого запустить таймер с частотой 44.1 кГц, в котором производить считывание данных с АЦП и запуск АЦП с максимальной скоростью так, чтобы к следующему входу в прерывание по таймеру АЦП закончило преобразование.

Контрольные теоретические вопросы

✓ Чем отличаются прерывания INT0 по фронту и по уровню? Какие регистры обслуживают внешние аппаратные прерывания?

✓ Что произойдет, если к кнопке INT0 не подключить резистор подтяжки?

✓ Укажите физические адреса векторов прерывания микроконтроллера ATmega16.

✓ Перечислите виды ШИМ 16-разрядного таймера/счетчика T/C1 микроконтроллера ATmega16.

Содержание отчета

Отчет должен содержать листинги отлаживаемых программ на языке С и код ассемблера, комментарии по ходу выполнения пунктов работы, примеры экранных форм (скриншоты), создаваемые по ходу выполнения работы (включая промежуточные действия и финальные задания), отображающие окна регистров и памяти, а также ответы на контрольные вопросы.

ЛИТЕРАТУРА

Основная литература

1. Бородин К. В. Микропроцессорные устройства и системы : учеб. пособие / К. В. Бородин. – Томск : ФДО, ТУСУР, 2016. – 137 с.

2. Русанов В. В. Микропроцессорные устройства и системы : учеб. пособие для вузов / В. В. Русанов, М. Ю. Шевелев. – Томск : Томск. ун-т систем упр. и радиоэлектроники, 2006. – 200 с. : ил.

3. Микропроцессорные системы : учеб. пособие для вузов / Е. К. Александров и др. ; под общ. ред. Д. В. Пузанкова. – СПб. : Политехника, 2002. – 935 с. : ил.

4. Шарапов А. В. Микропроцессорные устройства и системы : метод. указания к выполнению курсового проектирования / А. В. Шарапов. – Томск : Томск. ун-т систем упр. и радиоэлектроники, 2008. – 24 с.

5. Рождественский Д. А. Микропроцессорные устройства в системах управления : учеб. пособие / Д. А. Рождественский. – Томск : Том. межвуз. центр дистанционного образования, 2003. – 130 с.

6. Белов А. В. Конструирование устройств на микроконтроллерах / А. В. Белов. – СПб. : Наука и Техника, 2005. – 256 с. : ил.

Дополнительная литература

7. ATMega16. Datasheet. Atmel CO LTD. – USA, 2001.

8. Официальный сайт компании Atmel. – Режим доступа : http://www.atmel.com (дата обращения: 15.02.2016).